

**University of Science**  
**Viet Nam National University - Ho Chi Minh city**



**PROJECT REPORT**  
**KNAPSACK-PROBLEM**

**Introduction to Artificial Intelligence**

**Theory Lecturer** Nguyen Tien Huy  
**Practice Lecturer** Nguyen Tran Duy Minh

|                 |                      |          |
|-----------------|----------------------|----------|
| <b>Students</b> | Le Hoang Sang        | 21127158 |
|                 | Vo Thanh Tu          | 21127469 |
|                 | Nguyen Quoc Huy      | 21127511 |
|                 | Vo Pham Thanh Phuong | 21127677 |

# Table of content

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Preface</b>   | <b>2</b>  |
| <b>2</b> | <b>Assignment Plan</b>   | <b>2</b>  |
| <b>3</b> | <b>Completion level</b>  | <b>3</b>  |
| <b>4</b> | <b>Working with the Prolog tool</b>                                      | <b>3</b>  |
| 4.1      | The main features of the language . . . . .                              | 3         |
| 4.1.1    | Logic Programming . . . . .  | 3         |
| 4.1.2    | Facts . . . . .  | 4         |
| 4.1.3    | Variables . . . . .  | 4         |
| 4.1.4    | Unification . . . . .  | 4         |
| 4.1.5    | Backtracking . . . . .   | 4         |
| 4.1.6    | Cut . . . . .  | 5         |
| 4.2      | How to implement Prolog language on the studied tool . . . . .           | 5         |
| 4.2.1    | Installing SWI-Prolog . . . . .  | 6         |
| 4.2.2    | Running SWI-Prolog . . . . .   | 6         |
| 4.2.3    | Writing Prolog Programs . . . . .  | 8         |
| 4.2.4    | 5 illustrative examples . . . . .  | 9         |
| 4.3      | Solving deductive problems using Prolog language on SWI-Prolog tool. . . | 13        |
| <b>5</b> | <b>Build a Knowledge Base with Prolog</b>                                | <b>19</b> |
| 5.1      | Topic's introduction . . . . .   | 19        |
| 5.2      | Construct and Asking newly knowledge system . . . . .                    | 21        |
| 5.2.1    | Construct knowledge system . . . . .                                     | 21        |
| 5.3      | Asking newly constructed knowledge system . . . . .                      | 23        |
| <b>6</b> | <b>Implement logic deductive system in the programming language</b>      | <b>29</b> |
| <b>7</b> | <b>References</b>  | <b>31</b> |

# 1 Preface

## About our team:

Our group is group 3 according to the grouping list with 4 members.

| Student's ID | Assigned             |
|--------------|----------------------|
| 21127158     | Le Hoang Sang        |
| 21127469     | Vo Thanh Tu          |
| 21127511     | Nguyen Quoc Huy      |
| 21127677     | Vo Pham Thanh Phuong |

## About the project:

Our team researched the project, approached the Prolog language, gave specific examples, built a self-generated program as well as created a logical inference program by ourselves and compared it with the SWI-Prolog compiler through the available examples of the British Royal Family and examples that our team built.

# 2 Assignment Plan

| Task   | Assigned             | Status |
|--|----------------------|--------|
| Working with the Prolog tool                                 | Le Hoang Sang        | Done   |
| Build a Knowledge Base with Prolog                           | Nguyen Quoc Huy      | Done   |
| Build a Knowledge Base with Prolog                           | Vo Pham Thanh Phuong | Done   |
| Implement logic deductive system in the programming language | Vo Thanh Tu          | Done   |

### 3 Completion level

| No. | Criteria   | Scores | Status |
|-----|--|--------|--------|
| 1   | Working with the Prolog tool                                 | 40%    | Done   |
| 2   | Build a Knowledge Base with Prolog                           | 30%    | Done   |
| 3   | Implement logic deductive system in the programming language | 30%    | Done   |

## 4 Working with the Prolog tool

Prolog is a logic programming language widely used in artificial intelligence and natural language processing applications. It is based on the concept of first-order logic, which allows us to represent knowledge in a declarative manner using a set of logical rules and facts. In this report, we will discuss the main features of the Prolog language, with illustrative examples related to the knowledge of first-order logic.

### 4.1 The main features of the language

#### 4.1.1 Logic Programming

Prolog is a declarative language, where programs are expressed in terms of logical rules and facts. The language is based on the concept of Horn clauses, which are statements of the form "if A then B". These clauses can be used to represent both logical rules and facts. For example, we can represent the rule "All humans are mortal" as follows:

---

```
mortal(X) :- human(X).
```

---

This rule can be read as "X is mortal if X is human".

### 4.1.2 Facts

In Prolog, facts are statements that are true in the world we are modeling. They are represented as Horn clauses without a body. For example, we can represent the fact "John is a human" as follows:

---

```
1 human(john) .
```

---

### 4.1.3 Variables

Variables in Prolog are represented with an initial capital letter. They can be used to represent unknown values that satisfy certain conditions. For example, we can ask Prolog to find a mortal person as follows:

---

```
1 ?- mortal(X) .
```

---

Prolog will try to find a value for X that satisfies the mortal rule.

### 4.1.4 Unification

Unification is the process of finding values for variables that make two terms equal. In Prolog, this process is used to match queries with logical rules and facts. For example, if we query Prolog as follows:

---

```
1 ?- mortal(john) .
```

---

Prolog will try to find a rule or fact that unifies with the query. It will match the fact "human(john)" with the rule "mortal(X) :- human(X)", and infer that "john" is mortal.

### 4.1.5 Backtracking

Backtracking is the process of undoing choices made during a search for a solution. In Prolog, backtracking is used when there are multiple possible solutions for a query. For example, if we define the following rules:

---

```

1   parent(john, mary).
2   parent(john, tom).
3   parent(mary, ann).

```

---

We can ask Prolog to find all the children of "john" as follows:

---

```

1   ?- parent(john, X).

```

---

Prolog will first find "X = mary", but then backtracks and finds "X = tom". This process continues until all solutions are found.

#### 4.1.6 Cut

The cut operator (!) is used to prevent backtracking and commit to a certain solution. It is often used to optimize Prolog programs. For example, we can define the following rules to find the maximum value in a list:

---

```

1   max([X], X) :- !.
2   max([X|Xs], X) :- max(Xs, M), X >= M, !.
3   max([X|Xs], M) :- max(Xs, M), M > X.

```

---

The first rule states that the maximum of a list with one element is that element.

The second rule states that the maximum of a list with multiple elements is the first element if it is greater than or equal to the maximum of the rest of the list.

And the third rule states that the maximum of a list with multiple elements is the maximum of the rest of the list if it is greater than the first element. The cut operator is used to prevent backtracking in the second and third rules when a solution is found.

## 4.2 How to implement Prolog language on the studied tool

There are several Prolog programming environments available, such as SWI-Prolog, GNU Prolog, and YAP. In this report, we will discuss how to im-

plement the Prolog language on the SWI-Prolog environment, which is a popular and powerful Prolog implementation that provides a user-friendly development environment and a variety of built-in predicates and libraries.

### 4.2.1 Installing SWI-Prolog

To implement Prolog language on the SWI-Prolog environment, we first need to install the software. SWI-Prolog is available for Windows, MacOS, and Linux, and can be downloaded from the official website <http://www.swi-prolog.org/Download.html>.

### 4.2.2 Running SWI-Prolog

MacOS: if you use the Finder and double-click on SWI Prolog in Applications, depending on your security settings, you may get an error. If you do, try right-clicking (or control-clicking) on the application instead, and say “run it anyway”. You can also run it from the command line using the `swipl` executable in `/Applications/SWI-Prolog.app/Contents/MacOS`, assuming you installed it in the default location.

Windows: if you want to load a file, at least on the instructional machines you’ll either need to specify an absolute file path, or change the current working directory to the appropriate place. Here’s how to change the current working directory from within Prolog. If your files are in `Z:/CLC`, type this at the Prolog command prompt:

---

```
1      working_directory(X, 'Z:\\CLC').
```

---

(The double backslash is needed because a single backslash indicates some special character. And in case you’re wondering, `X` will be unified with the previous working directory.)

Some commands:

---

```
1      halt. % exit Prolog (short form: control-d)
2
```

---

```
3      consult(filename). % load the file filename.pl % (note the
      added .pl extension). If you need some more complex name (for
      example with a path), put it in single quotes, for example:
4      consult('/Users/schmertzkopf/squid.pl').
5
6      [filename]. % shorthand for consult
```

---



```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)
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).

R?- ['/Users/lehoangsang/coding/First-Order-Logic/RoyalFamily.pl']
|
true.

S?- grandparent(X, 'Mike Tindall').
false.

?- grandparent(X, 'Zara Phillips').
X = 'Queen Elizabeth II' ;
X = 'Prince Phillip' ;
false.

?- grandchild(Y, 'Queen Elizabeth II').
Y = 'Prince William' ;
Y = 'Prince Harry' ;
Y = 'Peter Phillips' ;
Y = 'Zara Phillips' ;
Y = 'Princess Beatrice' ;
Y = 'Princess Eugenie' ;
Y = 'James, Viscount Severn' ;
Y = 'Lady Louise Mountbatten-Windsor'.

?- |
```



### 4.2.3 Writing Prolog Programs

Prolog programs consist of a set of logical rules and facts. In SWI-Prolog, we can write Prolog programs directly in the interpreter, or we can save them in a file with a ".pl" extension and load them into the interpreter using the "consult" command. For example, we can save the following program in a file named "example.pl":

---

```
1      % Define some facts
2      father(john, tom).
3      father(john, mary).
4      father(tom, ann).
5      father(tom, bob).
6
7      % Define a rule
8      grandfather(X, Y) :- father(X, Z), father(Z, Y).
9
```

---

We can then load the program into the interpreter using the "consult" command as follows:

---

```
1      ?- consult('example.pl').
```

---

Once the program is loaded, we can use the "grandfather" rule to find all the grandfathers in the program as follows:

---

```
1      ?- grandfather(X, Y).
2      X = john,
3      Y = ann ;
4      X = john,
5      Y = bob ;
6      X = tom,
7      Y = ann ;
8      X = tom,
9      Y = bob ;
10     false.
```

---

This will return all the pairs of grandfathers and grandchildren in the program.

Moreover, we can use the "member" predicate to check if an element is a

member of a list and the "write" predicate to print a message to the console as follows:

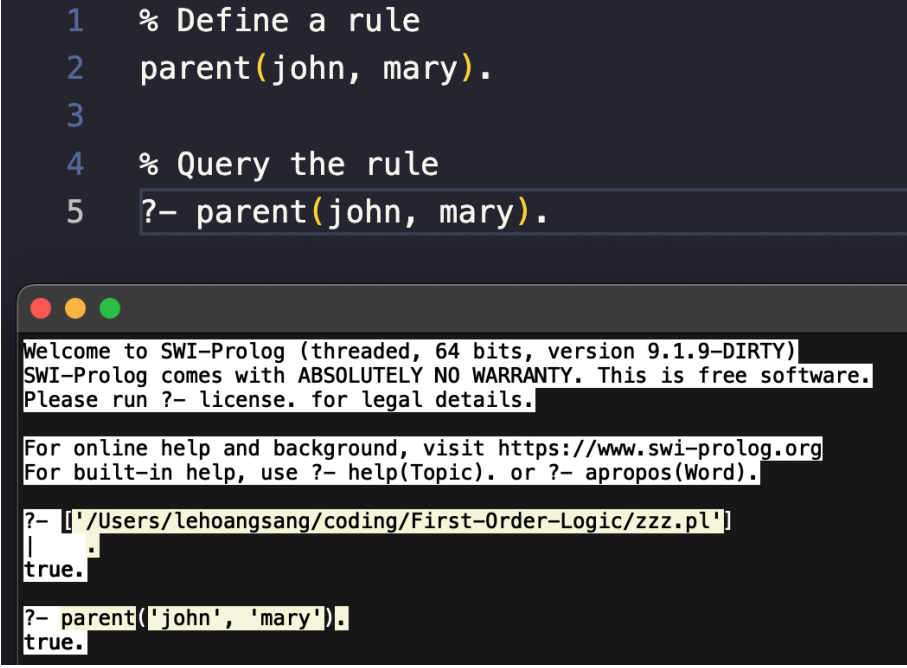
```
1      ?- member(2, [1,2,3]).
2      true.

1      ?- write('Hello, Mr.Huy and Mr. Minh!').
2      Hello, Mr.Huy and Mr.Minh!
3      true.
```

#### 4.2.4 5 illustrative examples

- Defining a rule and querying

```
1      % Define a rule
2      parent(john, mary).
3
4      % Query the rule
5      ?- parent(john, mary).
```



```
1      % Define a rule
2      parent(john, mary).
3
4      % Query the rule
5      ?- parent(john, mary).
```

Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)  
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).

```
?- ['Users/lehoangsang/coding/First-Order-Logic/zzz.pl']
|
true.

?- parent('john', 'mary').
true.
```

This example defines a rule that states "john" is the parent of "mary".

The query `parent(john, mary)` will return true.

- Implementing recursion

---

```

1      % Define a recursive rule to calculate factorial
2      factorial(0, 1).
3      factorial(N, Result) :-
4          N > 0,
5          N1 is N - 1,
6          factorial(N1, SubResult),
7          Result is N * SubResult.
8
9      % Query the factorial rule
10     ?- factorial(5, Result).
```

---

```

1  % Define a recursive rule to calculate factorial
2  factorial(0, 1).
3  factorial(N, Result) :-
4      N > 0,
5      N1 is N - 1,
6      factorial(N1, SubResult),
7      Result is N * SubResult.
8
9  % Query the factorial rule
10 ?- factorial(5, Result).
```

Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)  
 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).

?- ["/Users/lehoangsang/coding/First-Order-Logic/FiveExample/Ex02.pl"]  
 Warning: /Users/lehoangsang/coding/First-Order-Logic/FiveExample/Ex02.pl:10:  
 Warning: Singleton variables: [Result]  
 true.  
 ?- factorial(5, R).  
 R = 120

This example shows the implementation of a recursive rule to calculate the factorial of a number. The query `factorial(5, Result)` will return `Result = 120`.

- Working with lists

---

```

1      % Define a rule to find the length of a list
2      length([], 0).
3      length([_|T], N) :-
4          length(T, N1),
5          N is N1 + 1.
6
7      % Query the length rule
8      ?- length([1, 2, 3, 4, 5], Length).
```

---

```

1 % Define a rule to find the length of a list
2 length([], 0).
3 length([_|T], N) :-
4     length(T, N1),
5     N is N1 + 1.
6
7 % Query the length rule
8 ?- length([1, 2, 3, 4, 5], Length).

```

Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)  
 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).  
 ?- ["/Users/lehoangsang/coding/First-Order-Logic/FiveExample/Ex03.pl"]  
 ERROR: /Users/lehoangsang/coding/First-Order-Logic/FiveExample/Ex03.pl:2:  
 ERROR: No permission to modify static procedure 'length/2'.  
 ERROR: Defined at /Applications/SWI-Prolog.app/Contents/swipl/boot/init.pl:4184  
 ERROR: /Users/lehoangsang/coding/First-Order-Logic/FiveExample/Ex03.pl:3:  
 ERROR: No permission to modify static procedure 'length/2'.  
 ERROR: Defined at /Applications/SWI-Prolog.app/Contents/swipl/boot/init.pl:4184  
 Warning: /Users/lehoangsang/coding/First-Order-Logic/FiveExample/Ex03.pl:8:  
 Warning: Singleton variables: [Length]  
 true.  
 ?- Length([1, 2, 3, 4, 5], Length).  
 Length = 5.

This example demonstrates a rule to find the length of a list. The query `length([1, 2, 3, 4, 5], Length)` will return `Length = 5`.

- Using cut operator for optimization

```

1 % Define a rule to check if a number is even
2 even(X) :-
3     0 is X mod 2, !.
4
5 even(X) :-
6     1 is X mod 2.
7
8 % Query the even rule
9 ?- even(4).

```

```

1 % Define a rule to check if a number is even
2 even(X) :-
3     0 is X mod 2.
4
5 % Query the even rule
6 ?- even(4).

```

Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)  
 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).  
 ?- ["/Users/lehoangsang/coding/First-Order-Logic/FiveExample/Ex04.pl"]  
 true.  
 ?- even(3).  
 false.  
 ?- even(4).  
 true.

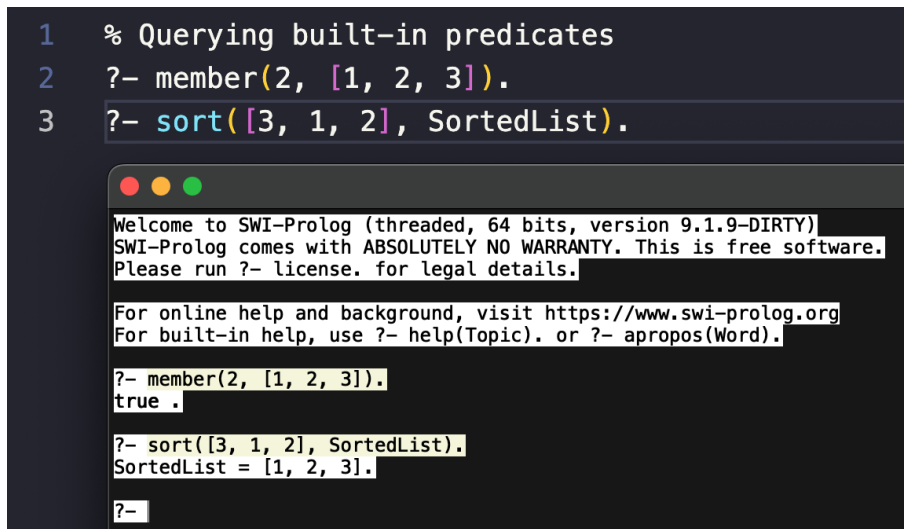
This example implements a rule to check if a number is even using the cut operator (!) to prevent unnecessary backtracking. The query `even(4)` will return `true`.

- Using built-in predicates

---

```
1  % Querying built-in predicates
2  ?- member(2, [1, 2, 3]).
3  ?- sort([3, 1, 2], SortedList).
```

---



```
1  % Querying built-in predicates
2  ?- member(2, [1, 2, 3]).
3  ?- sort([3, 1, 2], SortedList).
```

Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)  
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).

```
?- member(2, [1, 2, 3]).
true .

?- sort([3, 1, 2], SortedList).
SortedList = [1, 2, 3].

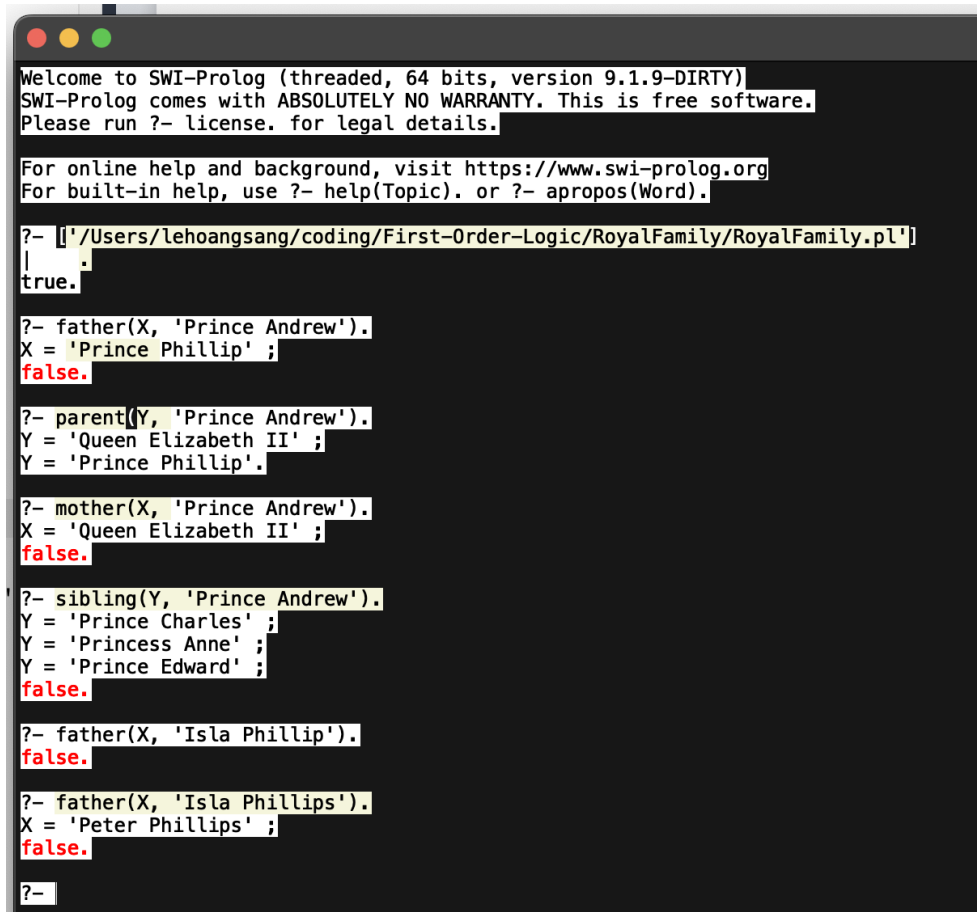
?-
```

These examples showcase the usage of built-in predicates. The first query checks if 2 is a member of the list [1, 2, 3], which will return true. The second query sorts the list [3, 1, 2], resulting in SortedList = [1, 2, 3].

### 4.3 Solving deductive problems using Prolog language on SWI-Prolog tool.

25 questions to ask the newly constructed knowledge system:

- Who is Prince Andrew's father?
- Who is Prince Andrew's parent?
- Who is Prince Andrew's mother?
- Who is Prince Andrew's sibling?
- Who is Isla Phillips' father?



```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)
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).

?- ['Users/lehoangsang/coding/First-Order-Logic/RoyalFamily/RoyalFamily.pl']
|
true.

?- father(X, 'Prince Andrew').
X = 'Prince Phillip' ;
false.

?- parent(Y, 'Prince Andrew').
Y = 'Queen Elizabeth II' ;
Y = 'Prince Phillip'.

?- mother(X, 'Prince Andrew').
X = 'Queen Elizabeth II' ;
false.

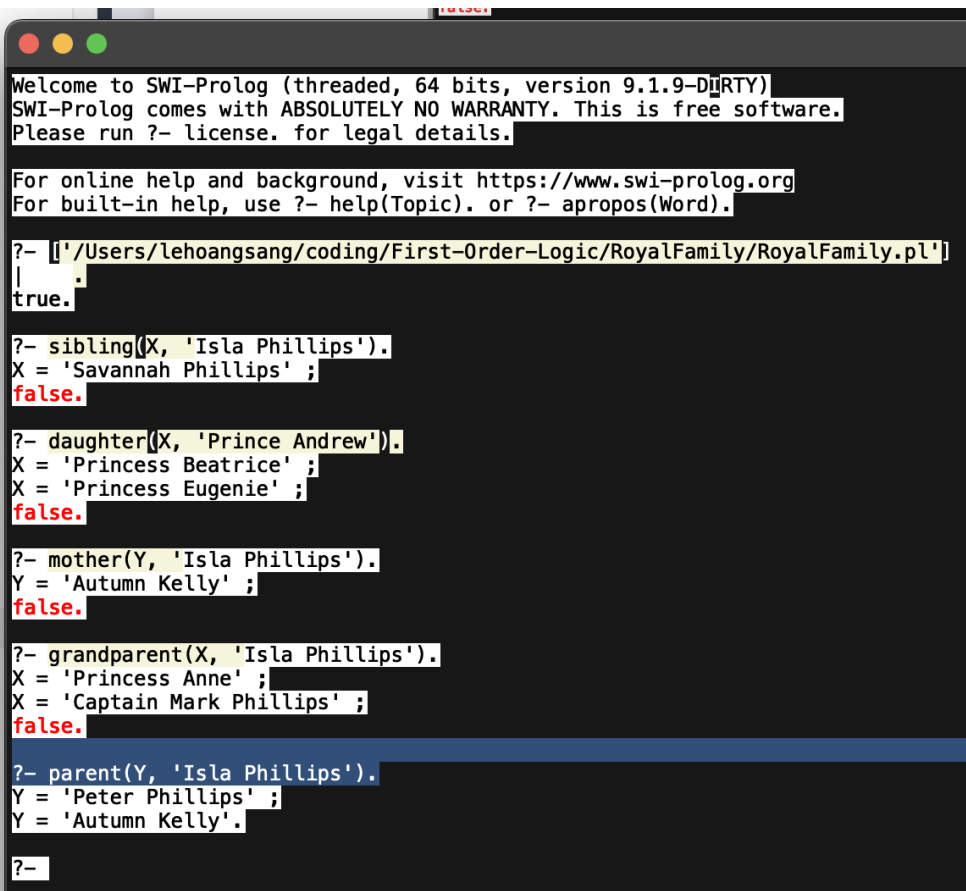
?- sibling(Y, 'Prince Andrew').
Y = 'Prince Charles' ;
Y = 'Princess Anne' ;
Y = 'Prince Edward' ;
false.

?- father(X, 'Isla Phillip').
false.

?- father(X, 'Isla Phillips').
X = 'Peter Phillips' ;
false.

?-
```

- Who is Isla Phillips' sibling?
- Who is Prince Andrew's daughter?
- Who is Isla Phillips' mother?
- Who is Isla Phillips' grandparent?
- Who is Isla Phillips' parent?



```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)
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).

?- ['./Users/lehoangsang/coding/First-Order-Logic/RoyalFamily/RoyalFamily.pl'].
true.

?- sibling(X, 'Isla Phillips').
X = 'Savannah Phillips' ;
false.

?- daughter(X, 'Prince Andrew').
X = 'Princess Beatrice' ;
X = 'Princess Eugenie' ;
false.

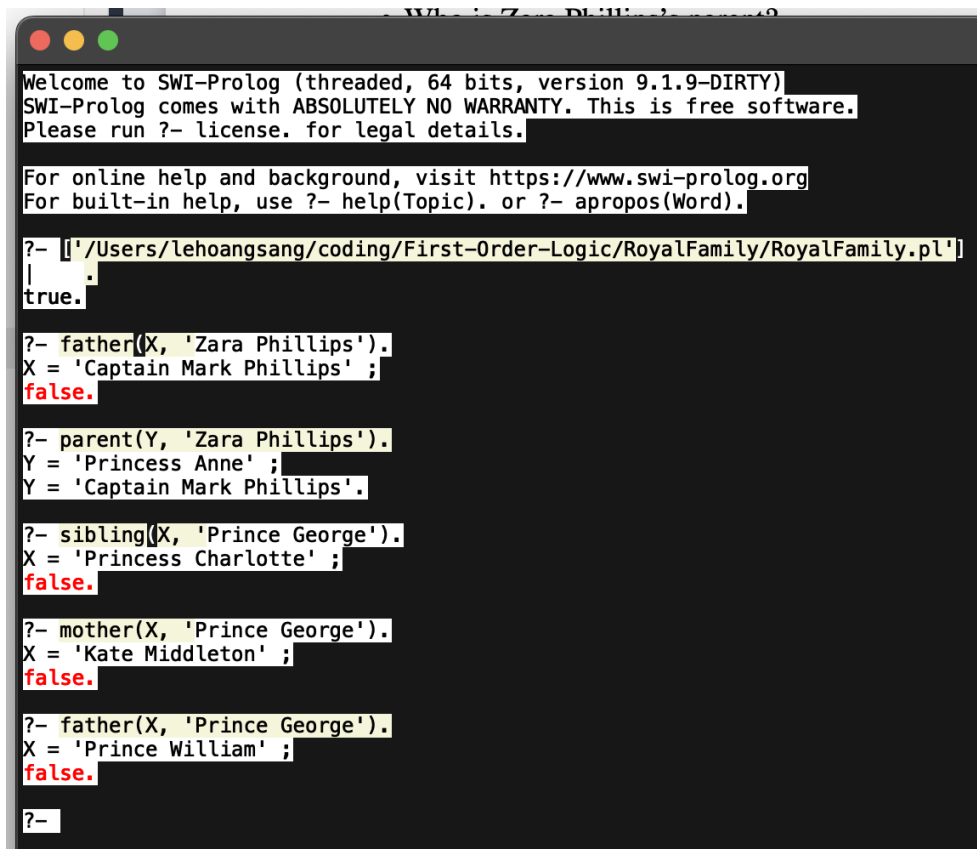
?- mother(Y, 'Isla Phillips').
Y = 'Autumn Kelly' ;
false.

?- grandparent(X, 'Isla Phillips').
X = 'Princess Anne' ;
X = 'Captain Mark Phillips' ;
false.

?- parent(Y, 'Isla Phillips').
Y = 'Peter Phillips' ;
Y = 'Autumn Kelly'.

?-
```

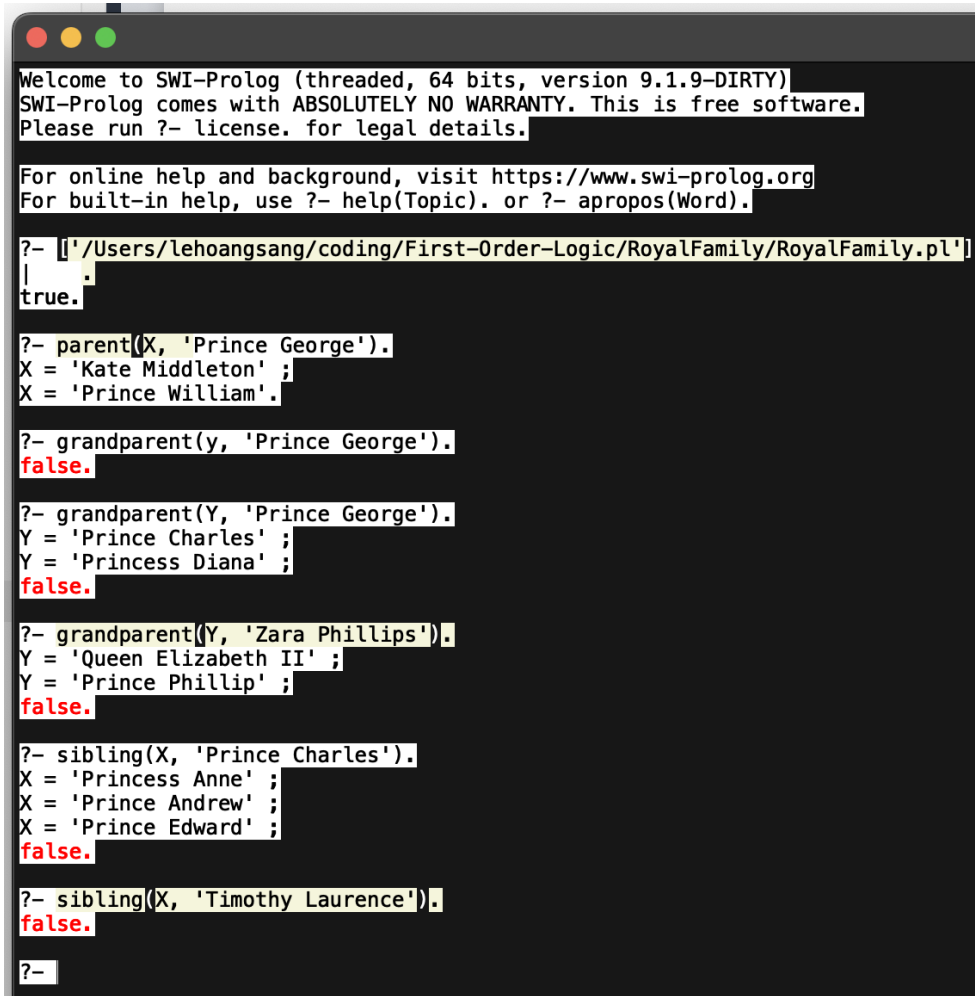
- Who is Zara Phillips' father?
- Who is Zara Phillips' parent?
- Who is Prince George's sibling?
- Who is Prince George's mother?
- Who is Prince George's father?



```
Who is Zara Phillips' parent?  
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)  
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).  
  
?- ['|'/Users/lehoangsang/coding/First-Order-Logic/RoyalFamily/RoyalFamily.pl'|  
|.].  
true.  
  
?- father(X, 'Zara Phillips').  
X = 'Captain Mark Phillips' ;  
false.  
  
?- parent(Y, 'Zara Phillips').  
Y = 'Princess Anne' ;  
Y = 'Captain Mark Phillips'.  
  
?- sibling(X, 'Prince George').  
X = 'Princess Charlotte' ;  
false.  
  
?- mother(X, 'Prince George').  
X = 'Kate Middleton' ;  
false.  
  
?- father(X, 'Prince George').  
X = 'Prince William' ;  
false.  
  
?-
```



- Who is Prince George's parent?
- Who is Prince George's grandparent?
- Who is Zara Phillips's grandparent?
- Who is Prince Charles' sibling?
- Who is Timothy Laurence's sibling?



```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)
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).

?- ['/Users/lehoangsang/coding/First-Order-Logic/RoyalFamily/RoyalFamily.pl']
|
true.

?- parent(X, 'Prince George').
X = 'Kate Middleton' ;
X = 'Prince William'.

?- grandparent(Y, 'Prince George').
false.

?- grandparent(Y, 'Prince George').
Y = 'Prince Charles' ;
Y = 'Princess Diana' ;
false.

?- grandparent(Y, 'Zara Phillips').
Y = 'Queen Elizabeth II' ;
Y = 'Prince Phillip' ;
false.

?- sibling(X, 'Prince Charles').
X = 'Princess Anne' ;
X = 'Prince Andrew' ;
X = 'Prince Edward' ;
false.

?- sibling(X, 'Timothy Laurence').
false.

?-
```

- Who is Zara Phillips' sibling?
- Who is Zara Phillips' mother?
- Who is Peter Phillips' child?
- Who is Prince Phillips' grandchild?
- Who is Prince Phillips's child?

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)
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).

?- ['Users/lehoangsang/coding/First-Order-Logic/RoyalFamily/RoyalFamily.pl']
|
true.

?- sibling(X, 'Zara Phillips').
X = 'Peter Phillips' ;
false.

?- mother(Y, 'Zara Phillips').
Y = 'Princess Anne' ;
false.

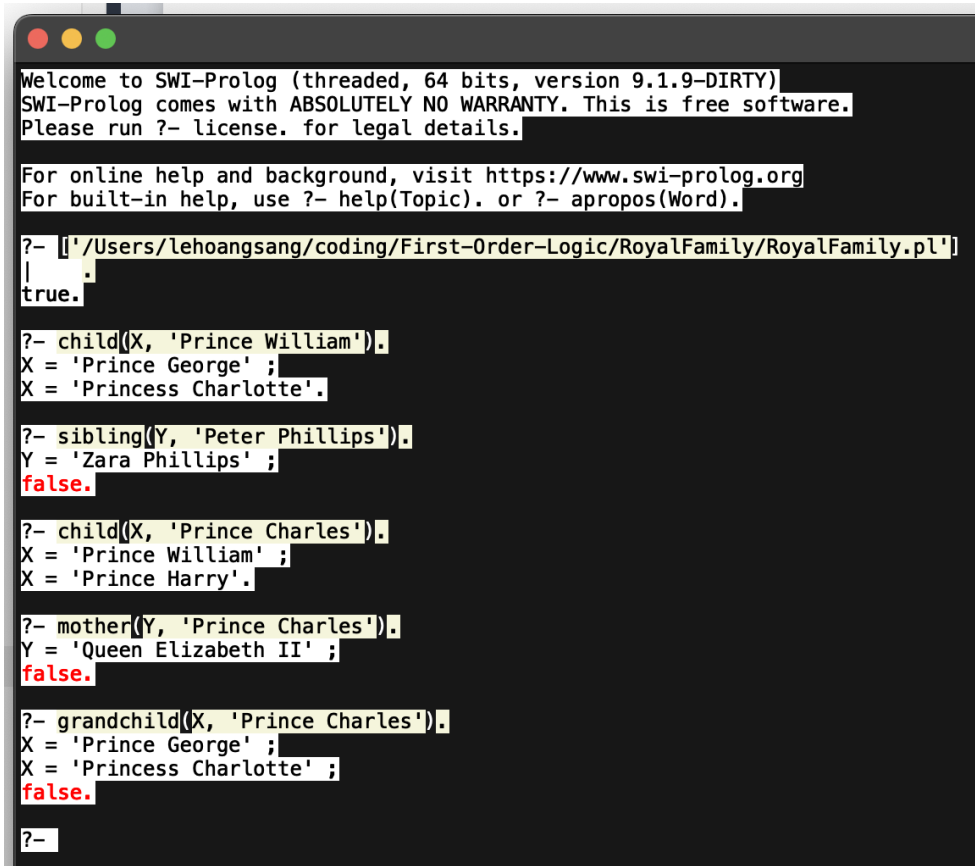
?- child(X, 'Peter Phillips').
X = 'Savannah Phillips' ;
X = 'Isla Phillips'.

?- grandchild(X, 'Prince Phillips').
false.

?- child(Y, 'Prince Phillips').
false.

?-
```

- Who is Prince William's child?
- Who is Peter Phillips's sibling?
- Who is Prince Charles's child?
- Who is Prince Charles's mother?
- Who is Prince Charles's grandchild?



```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.9-DIRTY)
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).

?- ['Users/lehoangsang/coding/First-Order-Logic/RoyalFamily/RoyalFamily.pl'].
|
true.

?- child(X, 'Prince William').
X = 'Prince George' ;
X = 'Princess Charlotte'.

?- sibling(Y, 'Peter Phillips').
Y = 'Zara Phillips' ;
false.

?- child(X, 'Prince Charles').
X = 'Prince William' ;
X = 'Prince Harry'.

?- mother(Y, 'Prince Charles').
Y = 'Queen Elizabeth II' ;
false.

?- grandchild(X, 'Prince Charles').
X = 'Prince George' ;
X = 'Princess Charlotte' ;
false.

?-
```

## 5 Build a Knowledge Base with Prolog

### 5.1 Topic's introduction

The topic of our group is about the Animal subspecies, with the highly researching and statistics from Kingdom, Class, Order, Family to Species. About over 90 creatures have been included and showed clearly in the sheet. However, due to the high concentration, some of these scientific names of the animals can be updated incorrectly, but it still ensures reasonable logic about the relationship. The below picture is the diagram of the relationship between the objects (Animals subspecies)

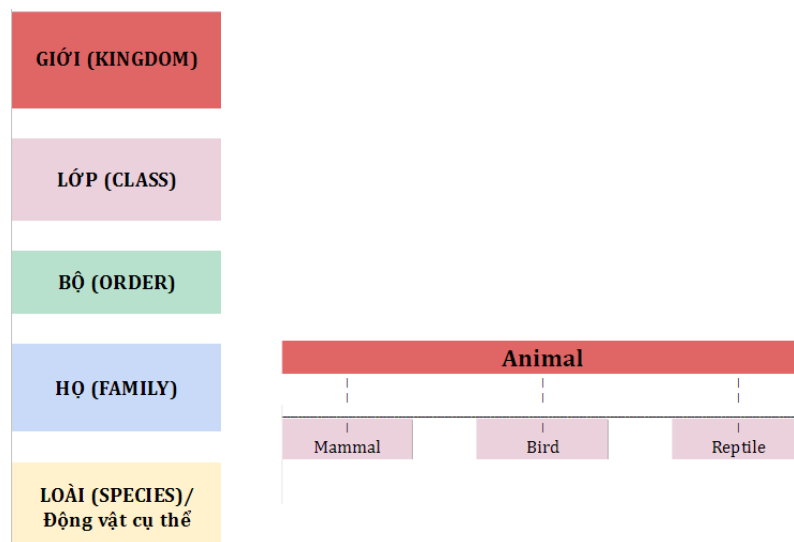


Figure 1: Overview of the diagram.

Due to the limitation of the image, we split the diagram into each Class subspecies images.

| LỚP (CLASS)                        | Mammal     |         |             |             |       |            |           |         |                 |         |
|------------------------------------|------------|---------|-------------|-------------|-------|------------|-----------|---------|-----------------|---------|
| BỘ (ORDER)                         | Primate    |         |             | Cetacean    |       |            | Carnivore |         |                 |         |
| HỌ (FAMILY)                        | Hominidae  |         | Hylobatidae | Delphinidae |       | Balaenidae | Ziphiidae | Felidae |                 | Canidae |
| LOÀI (SPECIES)/<br>Động vật cụ thể | Chimpanzee | Gibbon  |             | Dolphin     | Whale | Sea lion   |           | Panther | Panthera pardus | Wolf    |
|                                    | Gorilla    | Siamang |             |             |       |            |           | Tiger   |                 | Dog     |
|                                    | Human      |         |             |             |       |            |           | Lion    |                 |         |

Figure 2: Mammal class.

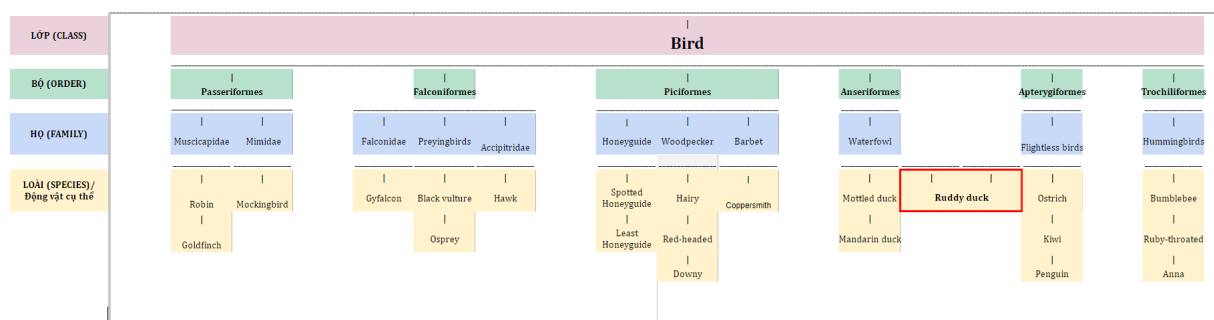


Figure 3: Bird class

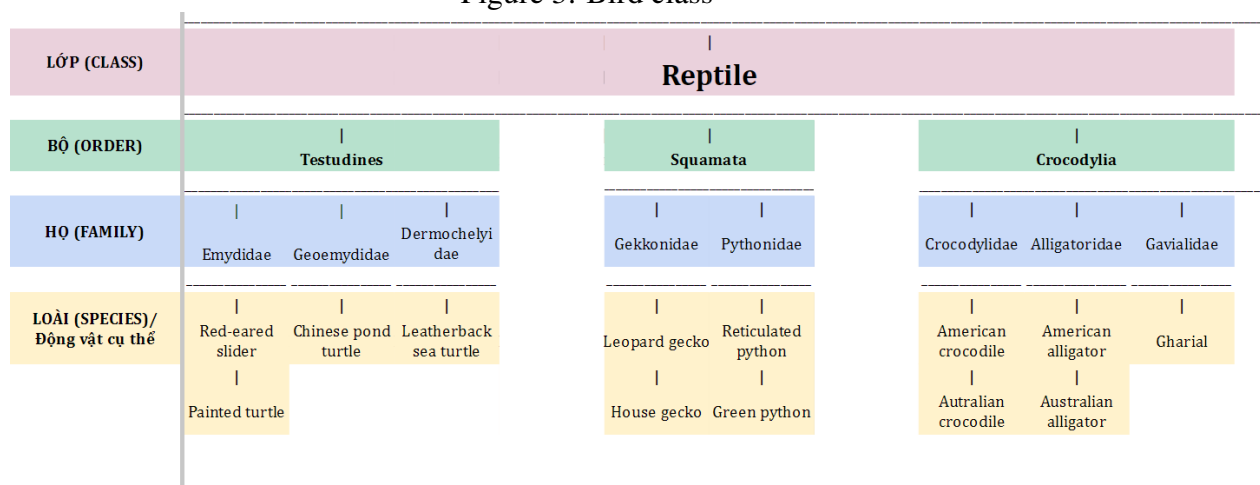


Figure 4: Reptile class.

## 5.2 Construct and Asking newly knowledge system

### 5.2.1 Construct knowledge system

To implement the knowledge base of this diagram, we have a base predicate to ensure the relationship between each animal and each branch. Beside, the addition predicates are also illustrated to determine the special relationship on each branch together or on each subspecies. The image below will show you clearly.

| No.                       | Knowledge base predicate       |
|---------------------------|--------------------------------|
| 1                         | NextLevel(Level1, Level2)      |
| <b>Addition predicate</b> |                                |
| 2                         | Family_of(Family, Animal)      |
| 3                         | Order_of(Order, Animal)        |
| 4                         | Class_of(Class, Animal)        |
| 5                         | Kingdom_of(Kingdom, Animal)    |
| 6                         | IsSameFamily(Animal1, Animal2) |
| 7                         | IsSameOrder(Animal1, Animal2)  |
| 8                         | IsSameClass(Animal1, Animal2)  |
| 9                         | BothFamily(Animal)             |
| 10                        | BothOrder(Animal)              |
| 11                        | BothClass(Animal)              |
| 12                        | IsSpecies(Animal)              |
| 13                        | IsKingdom(Animal)              |
| 14                        | IsClass(Animal)                |
| 15                        | IsOrder(Animal)                |
| 16                        | IsFamily(Animal)               |
| 17                        | CanCompete(Animal1, Animal2)   |
| 18                        | CanEat(Mammal, Bird)           |
| 19                        | CanFlight(Bird)                |

Figure 5: Knowledge base and addition predicates.

The base predicates were installed in turn through each branch, each relationship between animals, for example:

```

animalweb.pl
File Edit Browse Compile Prolog Pce Help
animalweb.pl
% Early definition
:- discontinuous nextlevel/2.

% General knowledge base
nextlevel('Animal', 'Mammal').
nextlevel('Animal', 'Bird').
nextlevel('Animal', 'Reptile').

% Mammal Class
nextlevel('Mammal', 'Primate').
nextlevel('Mammal', 'Cetacean').
nextlevel('Mammal', 'Carnivore').

nextlevel('Primate', 'Hominidae').
nextlevel('Primate', 'Hylobatidae').

nextlevel('Hominidae', 'Human').
nextlevel('Hominidae', 'Chimpanzee').
nextlevel('Hominidae', 'Gorilla').

nextlevel('Hylobatidae', 'Gibbon').
nextlevel('Hylobatidae', 'Siamang').

nextlevel('Cetacean', 'Delphinidae').
nextlevel('Cetacean', 'Balaenidae').
nextlevel('Cetacean', 'Ziphiidae').

nextlevel('Delphinidae', 'Dolphin').
nextlevel('Balaenidae', 'Whale').
nextlevel('Ziphiidae', 'Sea lion').

```

Figure 6: Implement base predicate.

On the other hand, the addition predicates are also installed by logic to show clearly the relationship between each argument, for example:

```

animalweb.pl
%Addition predicates.

%if the name in () is Species or Kingdom or Order or Class or Family.
isspecies(Animal) :- nextlevel('Animal', Class), nextlevel(Class, Order), nextlevel(Order, Family), nextlevel(Family, Animal).
iskingdom(Animal) :- Animal == 'Animal'.
isclass(Animal) :- nextlevel('Animal', Animal).
isorder(Animal) :- nextlevel('Animal', Class), nextlevel(Class, Animal).
isfamily(Animal) :- nextlevel('Animal', Class), nextlevel(Class, Order), nextlevel(Order, Animal).

%if that Animal is not belonged to Family, Order, Class, Kingdom.
family_of(Family, Animal) :- nextlevel(Family, Animal), isfamily(Family).
order_of(Order, Animal) :- (nextlevel(Order, Family), family_of(Family, Animal), isorder(Order)); (isorder(Order), nextlevel(Order, Animal)).
class_of(Class, Animal) :- (nextlevel(Class, Order), order_of(Order, Animal), isclass(Class)); (isclass(Class), nextlevel(Class, Animal)); (isclass(Class), nextlevel(Class, Order), nextlevel(Order, Animal)).
kingdom_of(Kingdom, Animal) :- iskingdom(Kingdom), (isclass(Animal); isfamily(Animal); isorder(Animal); isspecies(Animal)).

%if the Animal1 is same feature with the Animal2.
issamefamily(Animal1, Animal2) :- family_of(Family, Animal1), family_of(Family, Animal2).
issameorder(Animal1, Animal2) :- order_of(Order, Animal1), order_of(Order, Animal2).
issameclass(Animal1, Animal2) :- class_of(Class, Animal1), class_of(Class, Animal2).

%if the Animal has more than 1 feature.
bothfamily(Animal) :- family_of(Family1, Animal), family_of(Family2, Animal), Family1 \= Family2.
bothorder(Animal) :- order_of(Order1, Animal), order_of(Order2, Animal), Order1 \= Order2.
bothclass(Animal) :- class_of(Class1, Animal), class_of(Class2, Animal), Class1 \= Class2.

%if 2 animal can compete or eat each other.

```

Figure 7: Implement base predicate.

### 5.3 Asking newly constructed knowledge system

- Can a Red-headed Woodpecker fly?
- Is Ruddy duck in two families?
- Can a Tiger eat a Mottled duck?
- Is a Dolphin belonged to Reptile (Class)?
- Can a Gorilla compete with an American crocodile?
- Is Hylobatidae a family?
- Is Gibbon in same order with Human?
- Is Dog in two order?

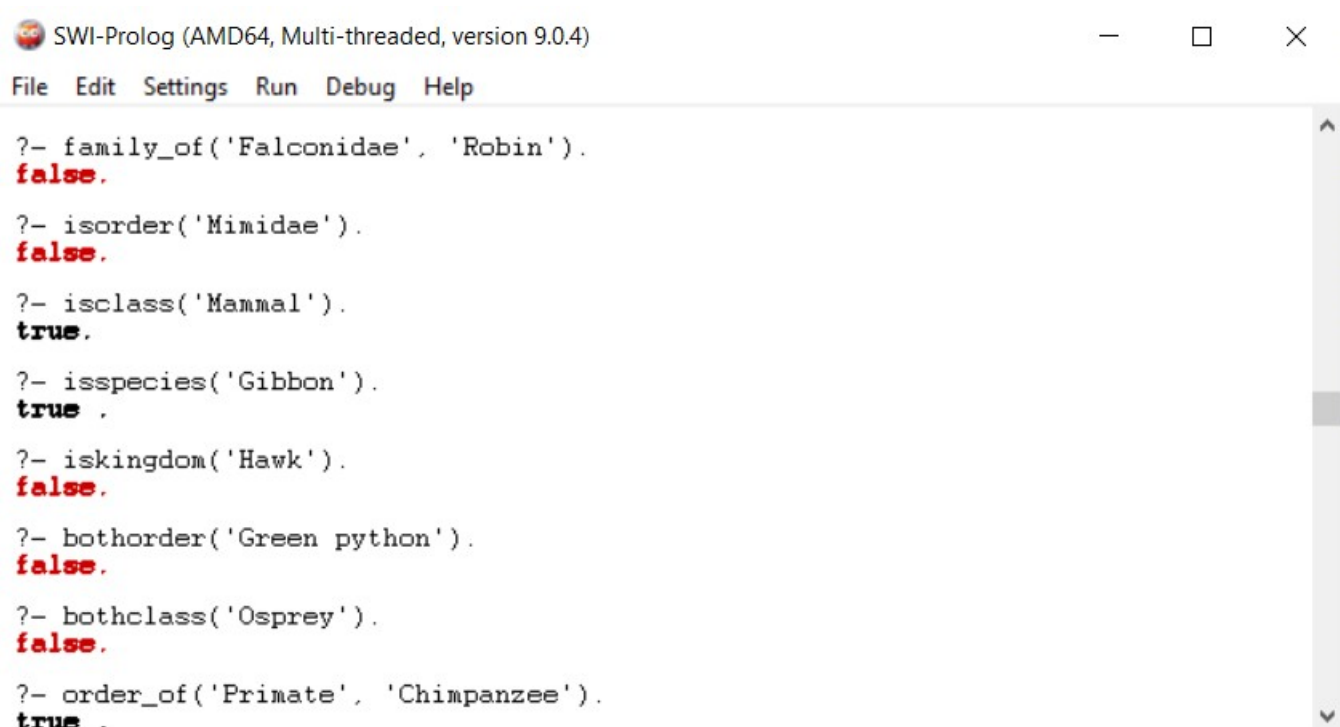




```
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
?- canflight('Red-headed').
true .
?- bothfamily('Ruddy duck').
true .
?- caneat('Tiger', 'Mottled duck').
true .
?- class_of('Reptile', 'Dolphin').
false.
?- cancompete('Gorilla', 'American crocodile').
true .
?- isfamily('Hylobatidae').
true .
?- issameorder('Gibbon', 'Human').
true .
?- bothorder('Dog').
false.
```

Figure 8: Questions toward the newly constructed knowledge system

- Is Robin belonged to Falconidae (Family)?
- Is Mimidae an order?
- Is Mammal a class?
- Is Gibbon a specie?
- Is Hawk a kingdom?
- Is Green python in two orders?
- Is Osprey in two classes?
- Is Chimpanzee belonged to Primate (Order)?



```
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

?- family_of('Falconidae', 'Robin').
false.

?- isorder('Mimidae').
false.

?- isclass('Mammal').
true.

?- isspecies('Gibbon').
true.

?- iskingdom('Hawk').
false.

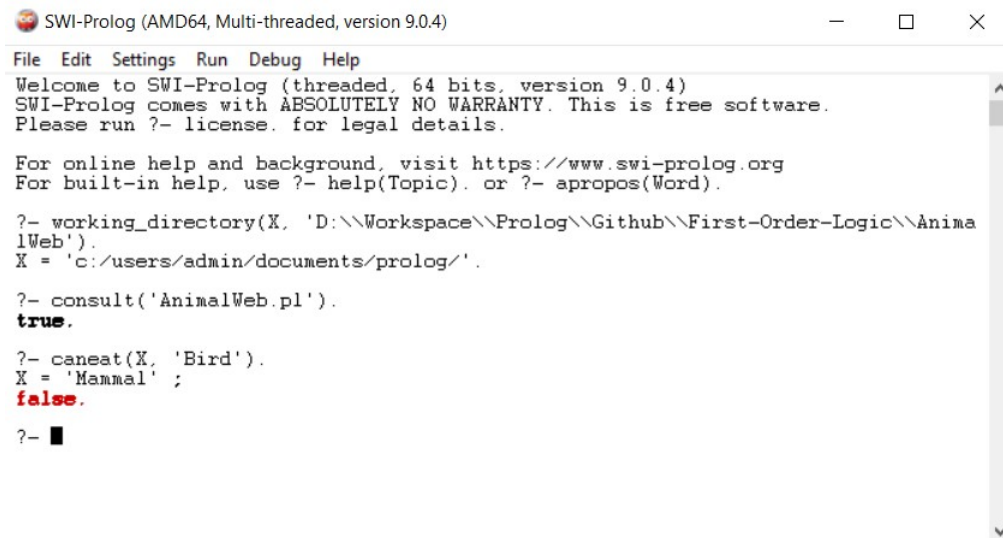
?- bothorder('Green python').
false.

?- bothclass('Osprey').
false.

?- order_of('Primate', 'Chimpanzee').
true.
```

Figure 9: Questions toward the newly constructed knowledge system

- What animals can eat Birds?



```
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).


?- working_directory(X, 'D:\\Workspace\\Prolog\\Github\\First-Order-Logic\\AnimalWeb').
X = 'c:/users/admin/documents/prolog/'.

?- consult('AnimalWeb.pl').
true.

?- caneat(X, 'Bird').
X = 'Mammal' ;
false.

?-
```

- What animals are in the same order with Downy woodpecker?



```
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

?- issameorder(X, 'Downy').
X = 'Spotted Honeyguide' ;
X = 'Least Honeyguide' ;
X = 'Red-headed' ;
X = 'Hairy' ;
X = 'Downy' ;
X = 'Coppersmith' ;
X = 'Honeyguide' ;
X = 'Woodpecker' ;
X = 'Barbet' ;
false.

?-
```

- What animals can compete with Human?

```

?- cancompete('Human', X).
X = 'Red-eared slider' ;
X = 'Painted turtle' ;
X = 'Chinese pond turtle' ;
X = 'Leatherback sea turtle' ;
X = 'Emydidae' ;
X = 'Geoemydidae' ;
X = 'Dermochelyidae' ;
X = 'Leopard gecko' ;
X = 'House gecko' ;
X = 'Reticulated python' ;
X = 'Green python' ;
X = 'Gekkomidae' ;
X = 'Pythonidae' ;
X = 'American crocodile' ;
X = 'Australian crocodile' ;
X = 'American alligator' ;
X = 'Australian alligator' ;
X = 'Gharial' ;
X = 'Crocodylidae' ;
X = 'Alligatoridae' ;
X = 'Gavialidae' ;
X = 'Testudines' ;
X = 'Squamata' ;
X = 'Crocodylia' ;
X = 'Emydidae' ;
X = 'Geoemydidae' ;
X = 'Dermochelyidae' ;
X = 'Gekkomidae' ;
X = 'Pythonidae' ;
X = 'Crocodylidae' ;
X = 'Alligatoridae' ;
X = 'Gavialidae' ;
X = 'Human' ;
X = 'Chimpanzee' ;
X = 'Gorilla' ;
X = 'Gibbon' ;
X = 'Siamang' ;
X = 'Hominidae' ;
X = 'Hylobatidae' ;
X = 'Dolphin' ;
X = 'Whale' ;
X = 'Sea lion' ;
X = 'Delphinidae' ;
X = 'Balaenidae' ;
X = 'Ziphiidae' ;
X = 'Tiger' ;
X = 'Lion' ;
X = 'Panther' ;
X = 'Panthera pardus' ;
X = 'Wolf' ;
X = 'Dog' ;
X = 'Panthera pardus' ;
X = 'Felidae' ;
X = 'Canidae' ;
X = 'Primate' ;
X = 'Cetacean' ;
X = 'Carnivore' ;
X = 'Hominidae' ;
X = 'Hylobatidae' ;

```

 SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)

File Edit Settings Run Debug Help

```

X = 'Dermochelyidae' ;
X = 'Leopard gecko' ;
X = 'House gecko' ;
X = 'Reticulated python' ;
X = 'Green python' ;
X = 'Gekkomidae' ;
X = 'Pythonidae' ;
X = 'American crocodile' ;
X = 'Australian crocodile' ;
X = 'American alligator' ;
X = 'Australian alligator' ;
X = 'Gharial' ;
X = 'Crocodylidae' ;
X = 'Alligatoridae' ;
X = 'Gavialidae' ;
X = 'Testudines' ;
X = 'Squamata' ;
X = 'Crocodylia' ;
X = 'Emydidae' ;
X = 'Geomydidae' ;
X = 'Dermochelyidae' ;
X = 'Gekkomidae' ;
X = 'Pythonidae' ;
X = 'Crocodylidae' ;
X = 'Alligatoridae' ;
X = 'Gavialidae' ;
X = 'Human' ;
X = 'Chimpanzee' ;
X = 'Gorilla' ;
X = 'Gibbon' ;
X = 'Siamang' ;
X = 'Hominidae' ;
X = 'Hylobatidae' ;
X = 'Dolphin' ;
X = 'Whale' ;
X = 'Sea lion' ;
X = 'Delphinidae' ;
X = 'Balaenidae' ;
X = 'Ziphiidae' ;
X = 'Tiger' ;
X = 'Lion' ;
X = 'Panther' ;
X = 'Panthera pardus' ;
X = 'Wolf' ;
X = 'Dog' ;
X = 'Panthera pardus' ;
X = 'Felidae' ;
X = 'Canidae' ;
X = 'Primate' ;
X = 'Cetacean' ;
X = 'Carnivore' ;
X = 'Hominidae' ;
X = 'Hylobatidae' ;
X = 'Delphinidae' ;
X = 'Balaenidae' ;
X = 'Ziphiidae' ;
X = 'Felidae' ;
X = 'Canidae' ;
false.

```

- What animals are in Hominidae order?

```
?- family_of('Hominidae', X).  
X = 'Human' ;  
X = 'Chimpanzee' ;  
X = 'Gorilla' ;  
false.  
?-
```

## 6 Implement logic deductive system in the programming language

Our application is written by C# with .NET 7.0 and C# 10.0 syntax.

### Features:

- Support fact terns, multi-fact terns.
- Support rules.
- Support conjunction terns.
- Unsupport negative terns.
- Unsupport record, list.
- Unsupport group terns by brackets.
- Unsupport disjunction terns.
- Unsupport anonymous variable,
- Support only comparision \= (difference) and == (equal). Deny other operators.

In this application, we using Forward Chaining algorithm for our reasoning systems. Because of the lack of optimization, our application is not as quick as SWI-Prolog system, but it have the same answers.

The images below show the questions asked by user and the answer of our self-constructed system.

```

?-nephew('Prince William', 'Prince Charles').
?-sibling('Princess Diana', 'Mike Tindall').
?-sibling(X, 'Mike Tindall').
?-sibling('Princess Diana', Y).
?-nephew('Princess Diana', Z).
?-grandchild('Queen Elizabeth II', 'Zara Phillips').
?- male('Mike Tindall').
?- divorced('Princess Anne', X).
?- parent('Queen Elizabeth II', 'Prince Charles').
?- child(X, 'Zara Phillips').
?- husband('Peter Phillips', X).
?- wife('Sarah Ferguson', 'Prince Andrew').
?- father(X, 'Princess Anne').
?- mother(X 'Prince harry').
?- daughter(X, 'Prince Edward').
?- son(X, 'Prince Edward').
?- grandfather('Prince Charles', 'Prince George').
?- grandmother(X, 'Peter Phillips').
?- granddaughter('Isla Phillips', 'Princess Anne').
?- sibling('Princess Beatrice', 'Princess Eugenie').
?- sibling(X, 'Lady Louise Mountbatten-Winsor').
?- brother(X, 'Princess Charlotte').
?- sister('Savannah Phillips', 'Isla Phillips').
?- aunt('Autumn Kelly', X).
?- uncle(X, 'Prince George').
?- niece('Mia Grace Tindall', 'Peter Phillips').
?- nephew('James, Viscount Severn', 'Prince Charles').
?- nephew(X,Y).

```

Figure 10: User's questions.

```

?-aunt('Autumn Kelly', X).
X = 'Mia Grace Tindall' ;
False.
Current fact: 450

?-uncle(X, 'Prince George').
X = 'Prince Harry' ;
False.
Current fact: 450

?-niece('Mia Grace Tindall', 'Peter Phillips').
True.
Current fact: 450

?-nephew('James, Viscount Severn', 'Prince Charles').
True.
Current fact: 450

?-nephew(X, Y).
X = 'Prince William'      ,
Y = 'Timothy Laurence'   ;
X = 'Prince William'      ,
Y = 'Prince Andrew'      ;
X = 'Prince William'      ,
Y = 'Prince Edward'      ;
X = 'Prince William'      ,
Y = 'Princess Anne'      ;
X = 'Prince William'      ,
Y = 'Sophie Rhys-jones'  ;

```

Figure 11: System's answer.

Beside, we also tested our self-constructed system with British Royal Family and our self-collected knowledge and it worked successfully. The extra fact found during running also counted and showed in detail.

## 7 References

- How to run SWI-Prolog.
- Compiling, Writing & Running Basic Prolog Code.
- Knowledge about the Animals Kingdom.
- Textbook TIF212 Prolog Tutorial 3.pdf

We appreciate your reading!