UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

Project 2: First-Order Logic

Subject: Introduction to Artificial Intelligence

Class: 21CLC03

Students: Trần Nguyên Huân – 21127050, Lê Huỳnh Phúc – 21127392

Nguyễn Hoàng Phúc – 21127671, Nguyễn Phú Trọng – 21127708

Instructors: Nguyễn Ngọc Thảo, Lê Ngọc Thành,

Nguyễn Trần Duy Minh, Nguyễn Hải Đăng

HO CHI MINH CITY – 2023

Table of Contents

I. Tasks assignment table, completion level

Full Name	Student Id	Tasks	Completion Level
Trần Nguyên Huân	21127050	- Write report and implement prolog program 1.1	100%
Lê Huỳnh Phúc	21127392	- Write report and implement prolog program 1.3	100%
Nguyễn Hoàng Phúc	21127671	- Write report and implement prolog program 1.2	100%
Nguyễn Phú Trọng	21127708	- Write report and implement prolog program 1.3	100%

II. Working with the Prolog tool

1. Introduction to Prolog

- Prolog is a logic programming language that was first developed in the 1970s, and it is based on the concept of first-order logic. Prolog stands for "Programming in Logic." It is widely used in artificial intelligence, natural language processing, expert systems, and other areas of computer science that require logical reasoning.
- Prolog programs consist of a series of statements that define relationships between objects or concepts. These statements are written in a logical notation that uses predicates, which are statements that assert the existence or non-existence of a particular relationship. Prolog then uses a search algorithm to find solutions to queries based on the rules and facts that have been defined in the program.
- Some notes on Prolog syntax:

English	Predicate Calculus	Prolog
If	→	:-
Not	~	Not
Or	V	;
And	^	,

- Variable names in Prolog start with an upper case. Predicate and function names often start with the lower case.
- All sentences in Prolog must end with a period (.).
- When defining a fact in Prolog, if you do not declare a specific variable, the default is the program understands that the variable has a value "for all".
  For example:

  likes(A, Kevin).   /* Everyone likes Kevin */
- Rule in Prolog has the form:

  left_hand_side :- right_hand_side .

  ➔ Interpreted as: left_hand_side ← right_hand_side. Where the expression left_hand_side catches forced to contain only 1 positive literal.
- Query in Prolog has the form:

  ?- query

❖ If the program finds the answer to the query or proves the query of user is correct, SWI-Prolog will display true or answers. In contrast, SWI-Prolog display false.

2. Main Features of Prolog

a. Declarative Programming

- Prolog is a declarative programming language, which means that you tell the computer what you want it to do, rather than how to do it. You define the rules and the relationships between the rules, and the computer figures out how to satisfy them.
- Example: Here is an example of a declarative rule in Prolog:

likes(john, pizza).

likes(mary, sushi).

likes(sue, sushi).

likes(john, curry).

➔ This code defines a relationship between people and food. It says that John likes pizza and curry, Mary likes sushi, and Sue likes sushi.

b. Logic Programming

- Prolog is a logic programming language, which means that it is based on the principles of formal logic. The language allows you to reason about relationships between objects and to express rules in a natural and intuitive way.
- Here is an example of a Prolog rule that uses logical reasoning:

```
likes(X, Y) :- pizza(Y), italian(X).
```

➔ This code defines a rule that says that X likes Y if Y is pizza and X is Italian. In other words, if someone is Italian, they are more likely to like pizza.

c. Backtracking

- Prolog is designed to support backtracking, which means that the program can "undo" its choices and try alternative paths if it fails to satisfy a condition.
- Here is an example of a Prolog rule that uses backtracking:

```
parent(john, mary).
parent(john, sue).
parent(jane, mary).
parent(jane, sue).
sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.
```

➔ This code defines a relationship between people and their siblings. It says that X is a sibling of Y if they have the same parent. If you ask Prolog who the siblings of Mary are, it will first find John and then backtrack to find Jane.

d. Unification

- Prolog uses a process called unification to match patterns and variables. Unification is the process of finding values for variables that satisfy a set of constraints.
- Here is an example of a Prolog rule that uses unification:

```
color(red, green, blue) :- red, green, blue.
```

➔ This code defines a color as a combination of red, green, and blue. It uses unification to ensure that all three colors are present.

e. Recursion

- Prolog supports recursion, which means that a rule can call itself repeatedly until a condition is met.
- Here is an example of a Prolog rule that uses recursion:

```
factorial(0, 1).
factorial(N, F) :- N > 0, N1 is N-1, factorial(N1, F1), F is N * F1.
```

➔ This code defines a rule for calculating the factorial of a number. It uses recursion to repeatedly call itself with smaller values of N until N is 0, at which point it returns the result.

- Conclusion: Prolog is a powerful and flexible language that is based on the principles of formal logic. It is particularly well-suited for applications that require logical reasoning, such as artificial intelligence and expert systems. The language's declarative style, logical programming features, backtracking, unification, and support for recursion make it a valuable tool for many applications.

3. Implement Prolog language on SWI-Prolog environment

a. Introduction to SWI-Prolog

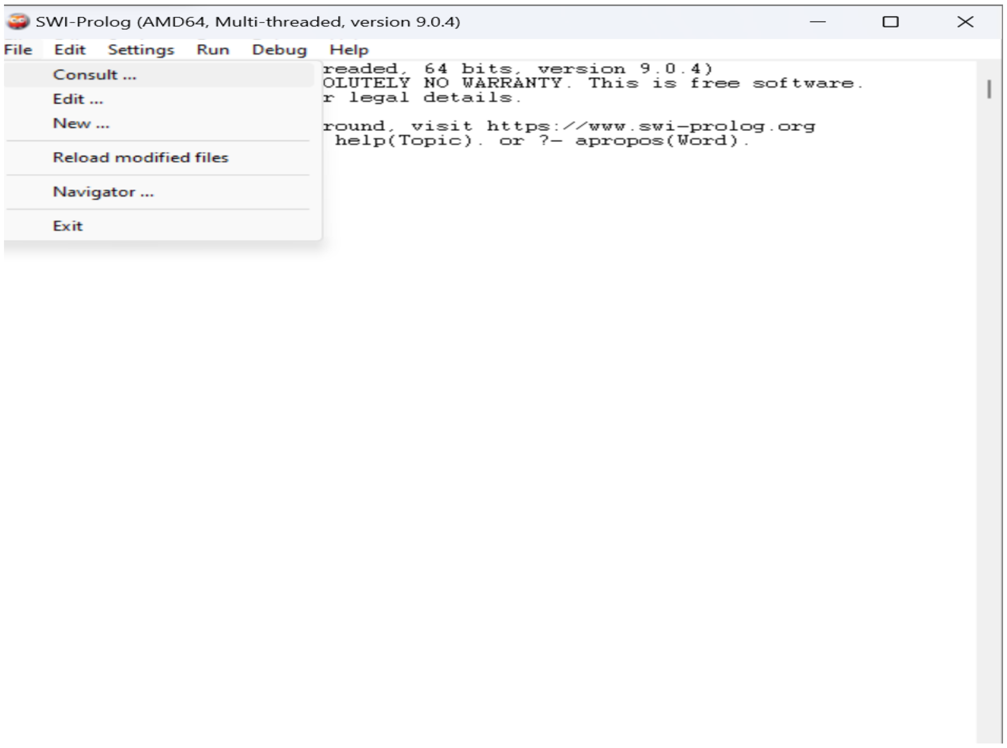
- SWI-Prolog is a popular and widely used programming environment for the Prolog language. It provides a rich set of tools and features for developing and debugging Prolog programs, as well as comprehensive documentation and an active community.

b. Implementation:

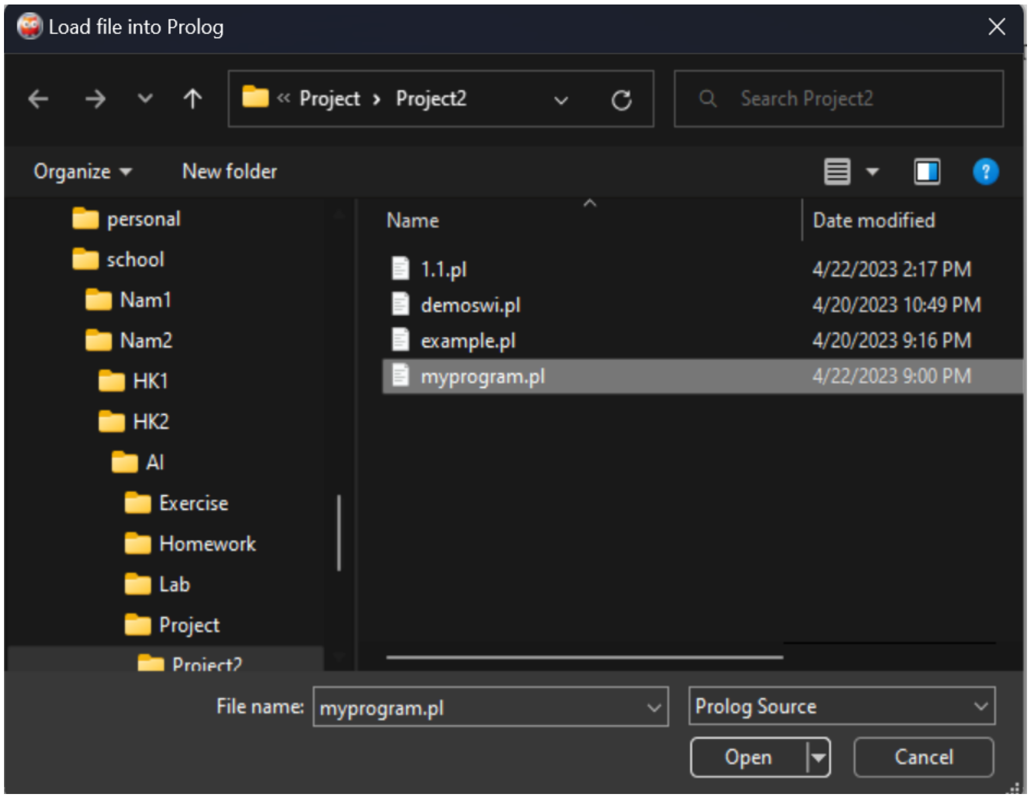
- To implement Prolog on SWI-Prolog, follow these steps:

1) Download and install SWI-Prolog from the official website: http://www.swi-prolog.org/Download.html
2) Launch SWI-Prolog and open a new file by selecting File > New from the menu bar.
3) Write your Prolog code in the file, following the syntax and rules of the language.
4) Save the file with a .pl extension, for example, myprogram.pl.

5) Compile and run the program by selecting File > Consult from the menu bar, then selecting your program file.


SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help
Consult ...
Edit ...
New ...
Reload modified files
Navigator ...
Exit

```
readed, 64 bits, version 9.0.4)
OLUTELY NO WARRANTY. This is free software.
r legal details.

round, visit https://www.swi-prolog.org
 help(Topic). or ?- apropos(Word).
```



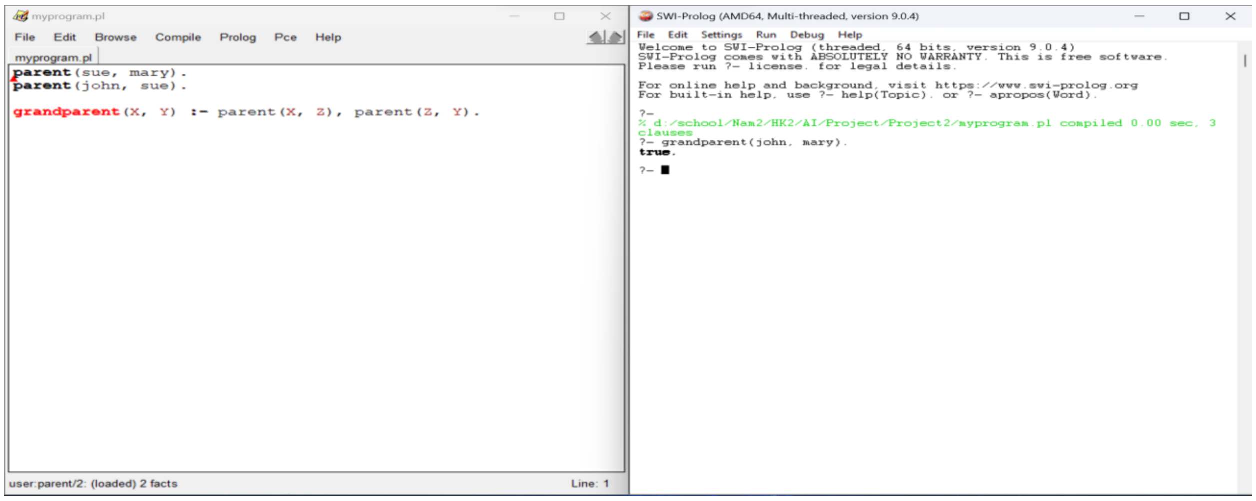
Name	Date modified
1.1.pl	4/22/2023 2:17 PM
demoswi.pl	4/20/2023 10:49 PM
example.pl	4/20/2023 9:16 PM
myprogram.pl	4/22/2023 9:00 PM

File name: myprogram.pl

6) You can interact with the program by entering queries into the Prolog interpreter, which is located at the bottom of the SWI-Prolog window.

c. Illustrative Examples

1) Defining relationships between people:

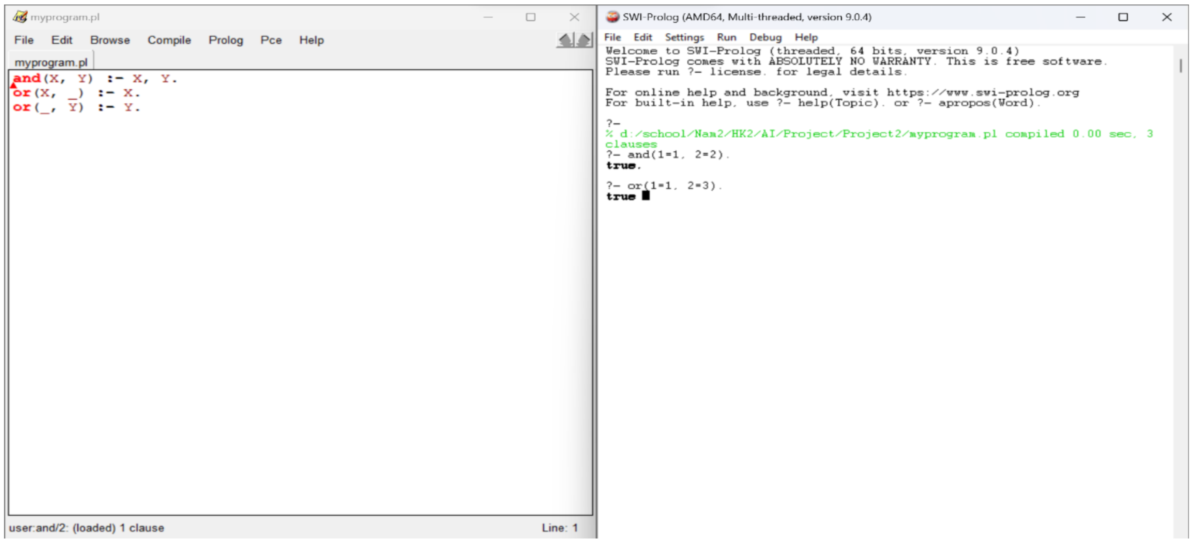

myprogram.pl
File Edit Browse Compile Prolog Pce Help
myprogram.pl
user:parent/2: (loaded) 2 facts
Line: 1
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

```prolog
parent(sue, mary).
parent(john, sue).

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
```

```
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).

?-
% d:/school/Nam2/HK2/AI/Project/Project2/myprogram.pl compiled 0.00 sec, 3
clauses
?- grandparent(john, mary).
true.

?-
```

➔ This program defines relationships between people and uses the grandparent rule to find out if John is Mary's grandparent.

2) Implementing logical operators:

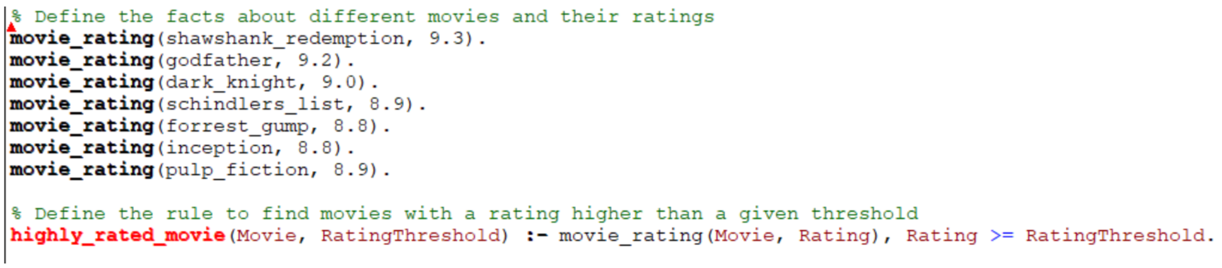

myprogram.pl
File Edit Browse Compile Prolog Pce Help
myprogram.pl
user:and/2: (loaded) 1 clause
Line: 1
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

```prolog
and(X, Y) :- X, Y.
or(X, _) :- X.
or(_, Y) :- Y.
```

```
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).

?-
% d:/school/Nam2/HK2/AI/Project/Project2/myprogram.pl compiled 0.00 sec, 3
clauses
?- and(1=1, 2=2).
true.

?- or(1=1, 2=3).
true
```

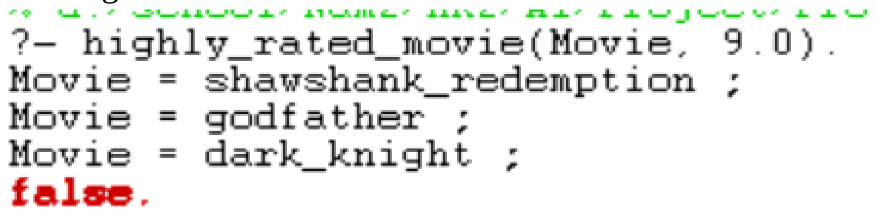
➔ This program defines logical operators in Prolog and uses them to evaluate two expressions.

3) Implementing a knowledge base:



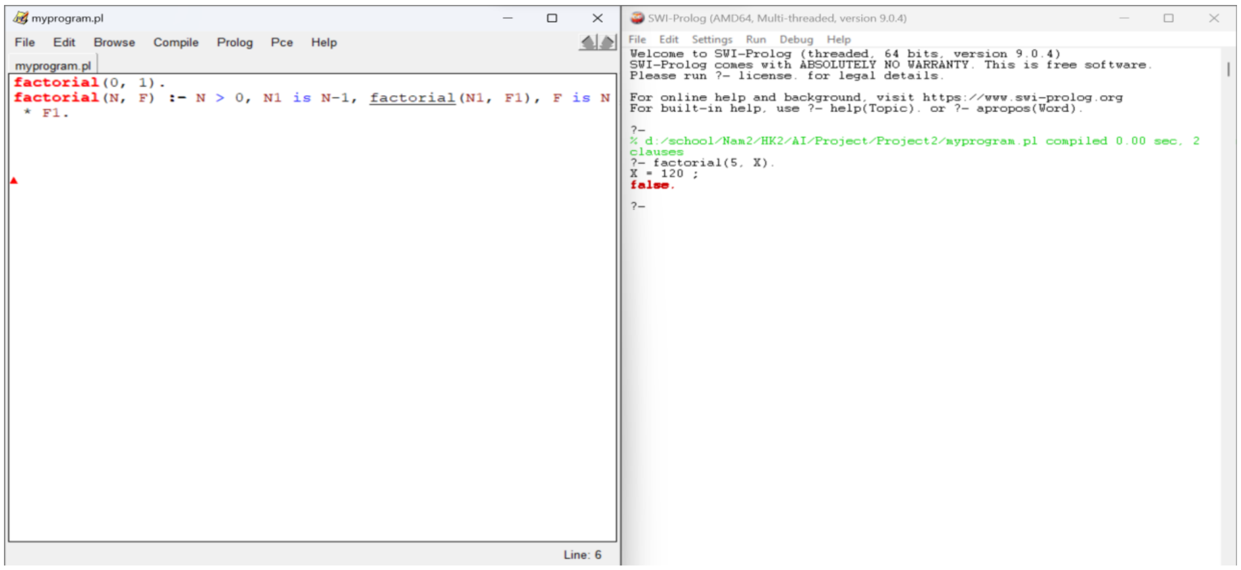
```
% Define the facts about different movies and their ratings
movie_rating(shawshank_redemption, 9.3).
movie_rating(godfather, 9.2).
movie_rating(dark_knight, 9.0).
movie_rating(schindlers_list, 8.9).
movie_rating(forrest_gump, 8.8).
movie_rating(inception, 8.8).
movie_rating(pulp_fiction, 8.9).

% Define the rule to find movies with a rating higher than a given threshold
highly_rated_movie(Movie, RatingThreshold) :- movie_rating(Movie, Rating), Rating >= RatingThreshold.
```

- In this knowledge base, we define facts about different movies and their ratings. The rule "highly_rated_movie" is then defined to find all movies with a rating higher than a given threshold.
- For example, if we query highly_rated_movie(Movie, 9.0)., we will get the following result:



```
?- highly_rated_movie(Movie, 9.0).
Movie = shawshank_redemption ;
Movie = godfather ;
Movie = dark_knight ;
false.
```

4) Implementing a recursive function:



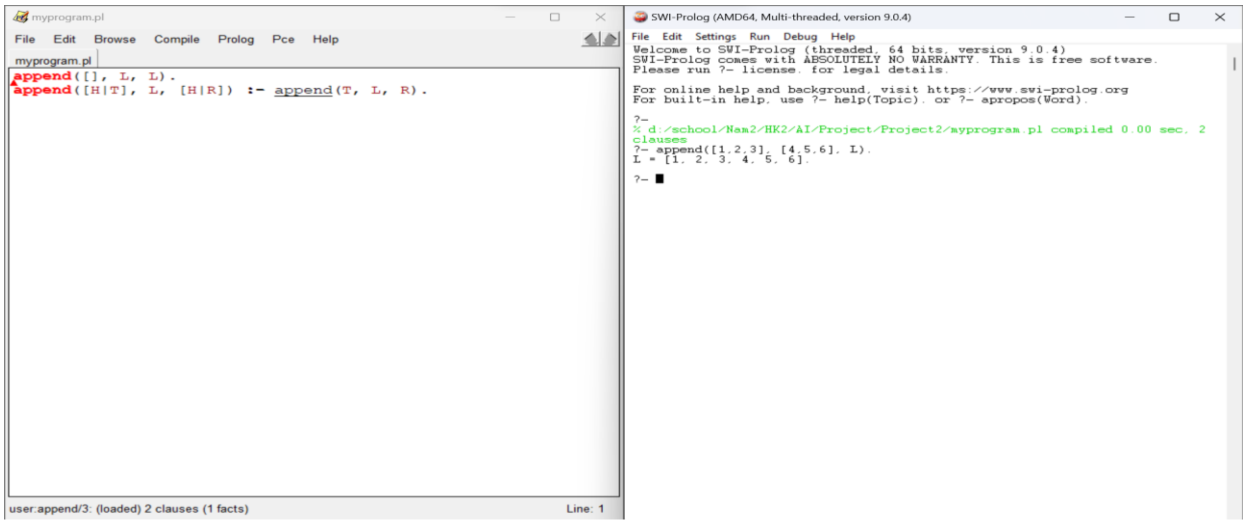
myprogram.pl
File Edit Browse Compile Prolog Pce Help
myprogram.pl
Line: 6
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

```
factorial(0, 1).
factorial(N, F) :- N > 0, N1 is N-1, factorial(N1, F1), F is N
 * F1.
```

```
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).

?-
% d:/school/Nam2/HK2/AI/Project/Project2/myprogram.pl compiled 0.00 sec, 2
clauses
?- factorial(5, X).
X = 120 ;
false.

?-
```

➔ This program defines a recursive function to calculate the factorial of a number and uses it to find the factorial of 5.

5) Implementing a list manipulation:

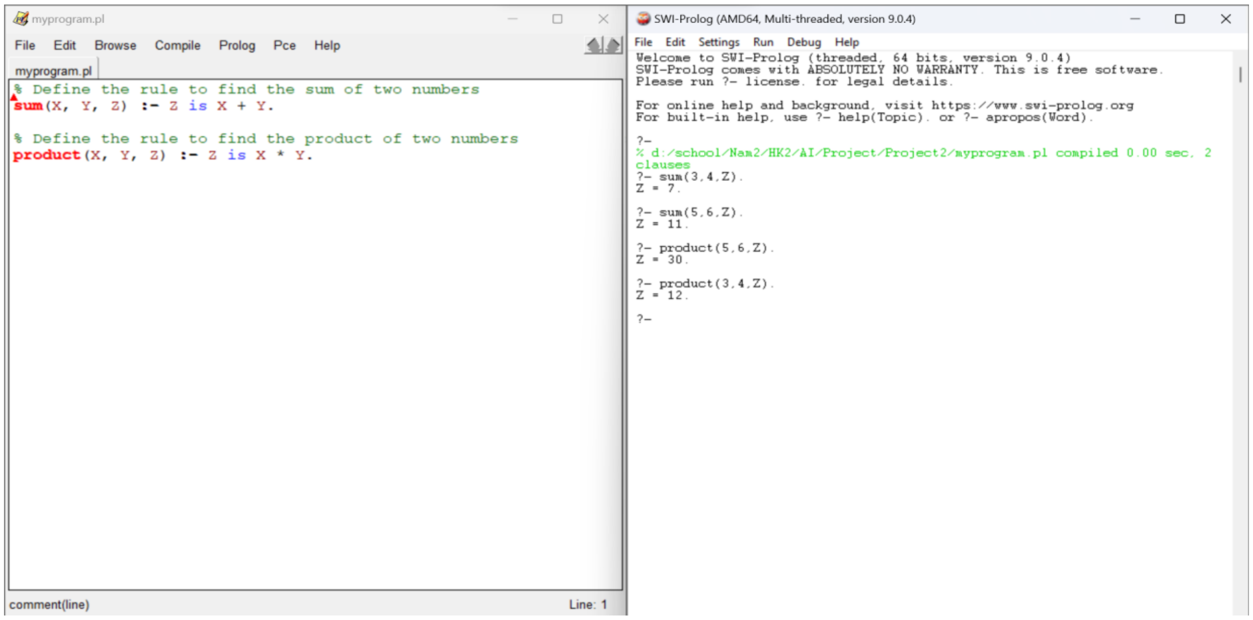

myprogram.pl
File Edit Browse Compile Prolog Pce Help
myprogram.pl
user:append/3: (loaded) 2 clauses (1 facts)
Line: 1
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

```
append([], L, L).
append([H|T], L, [H|R]) :- append(T, L, R).
```

```
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).

?-
% d:/school/Nam2/HK2/AI/Project/Project2/myprogram.pl compiled 0.00 sec, 2
clauses
?- append([1,2,3], [4,5,6], L).
L = [1, 2, 3, 4, 5, 6].

?- 
```

➔ This program defines a rule to append two lists and uses it to append two lists of numbers.

6) Mathematical operations:


myprogram.pl
File Edit Browse Compile Prolog Pce Help
myprogram.pl
comment(line)
Line: 1
SWI-Prolog (AMD64, Multi-threaded, version 9.0.4)
File Edit Settings Run Debug Help

```
% Define the rule to find the sum of two numbers
sum(X, Y, Z) :- Z is X + Y.

% Define the rule to find the product of two numbers
product(X, Y, Z) :- Z is X * Y.
```

```
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).

?-
% d:/school/Nam2/HK2/AI/Project/Project2/myprogram.pl compiled 0.00 sec, 2
clauses
?- sum(3,4,Z).
Z = 7.

?- sum(5,6,Z).
Z = 11.

?- product(5,6,Z).
Z = 30.

?- product(3,4,Z).
Z = 12.

?-
```

➔ In this example, we define rules to find the sum and product of two numbers, where the "is" keyword is used to evaluate the expression and bind the result to a variable.

4. Solving deductive problems using Prolog (SWI-Prolog)

- Build a family tree of the ***British Royal family***.
  a. Definition and inference of predicate

- **Group of pre-defined predicates**: include parent(Parent,Child), male(Person), female(Person), divorced(Person, Person) and married(Person, Person).
  **Notes:* With this group of predicates, note that the marital status relationship needs to define both facts: married(X,Y) and married(Y,X); divorced(X,Y) and divorced(Y,X).
- **Group of inferred predicates**:
  - **husband(Person, Wife):** Person is male and is married to the Wife object.
  - **wife(Person, Husband):** Person is female and is married to the Husband object.
  - **father(Parent, Child):** Parent is the male and parent of the Child object.
  - **mother(Parent, Child):** Parent is female and is the parent of the Child object.
  - **child(Child, Parent)**: child whose parent is a Parent object.
  - **son(Child, Parent):** Child is a male and is a child of the Parent object.
  - **daughter(Child, Parent):** Child is female and is the child of the Parent object.
  - **grandparent(GP, GC):** GP is the parent of the parent of the GC object.
  - **grandmother(GM, GC):** GP is the female and grandparent of the GC subject.
  - **grandfather(GF, GC):** GP is the male and grandparent of the GC object.
  - **grandchild(GC, GP):** GC has a grandparent as a GP object.
  - **grandson(GS, GP):** GS is the male and grandchild of the GP object.
  - **granddaughter(GD, GP):** GD is female and grandchild of GP.
  - **sibling(Person1, Person2):** Person1 is different from Person2, the parent of Person1 and Person2 is the same.
  - **brother(Person, Sibling):** Person is male and is Sibling's sibling.
  - **sister(Person, Sibling):** Person is female and is Sibling's sibling.

- **aunt(Person, NieceNephew):** Person is a female, Person is the sibling of the parent of NieceNephew or Person is a female, Person is married to the sibling of the parent of NieceNephew.
- **uncle(Person, NieceNephew):** Person is a male, Person is the sibling of the parent of NieceNephew or Person is a male, Person is married to the sibling of the parent of NieceNephew.
- **niece(Person, AuntUncle):** Person is female and has an aunt/uncle of AuntUncle.
- **nephew(Person, AuntUncle):** Person is male and has an aunt/uncle of AuntUncle.

b. Set of query questions

* The order of the set of questions and the answers are shown in file "test_dataset.txt".

1) Who are Queen Elizabeth's children?
2) Who is Prince Charles' mother?
3) Who are Prince William's parents?
4) Who are Prince Philip's grandchildren?
5) Is Kate Middleton the wife of Prince William?
6) Who is Prince Harry's aunt?
7) Who are Prince George's parents?
8) Who are Princess Beatrice'siblings?
9) Is Peter Phillips brother of Zara Phillips?
10) Who are Prince Edward's nieces?
11) Is Prince Phillip grandfather of Mia Grace Tindall?
12) Who is Prince Charles's granddaughter?
13) Who are Prince Andrew's nephew?
14) Who is Peter Phillips's grandfather?
15) Is James Viscount Severn son of Sophie Rhys-jones?
16) Who is Princess Anne's husband?
17) Who is Camilla Parker Bowles' uncle?
18) Who are Peter Phillips's daughters?
19) Who are Queen Elizabeth's grandson?
20) Is Savannah Phillips sister of Isla Phillips?
21) Who is grandmother of Lady Louise Mountbatten Windsor?

III. Build a Knowledge Base with Prolog

1. Definition

We will create a knowledge base of nations and cities, and define a set of predicates that relate to the countries, their regions, and their cities. The predicates used in the knowledge base are as follows:

- **region(Region, Continent)**: Connects a region with a continent.
- **country(Country, Region)**: Connects a country with a region.
- **city(City, Country)**: Connects a city with a country.
- **capital(City, Country)**: Identifies a city as the capital of a specific country.
- **neighbor(Country1, Country2)**: Defines neighboring countries.

*Inferred predicates:

- **neighbor(Country1, Country2, Country3) :- neighbor(Country1, Country2), neighbor(Country1, Country3).**: Country1 is neighbor to both Country2 and Country3.
- **city_region(City, Region) :- city(City, Country), country(Country, Region).:** Connects a city with a region
- **continent(Country, Continent) :- country(Country, Region), region(Region, Continent). :** Connects a country with a continent
- **in_continent(City, Continent) :- city(City, Country), country(Country, Region), region(Region, Continent). :** Connects a city with a continent.
- **in_same_continent(City1, City2) :- in_continent(City1, Continent), in_continent(City2, Continent). :** Checks if two cities are in the same continent.
- **capital_continent(Capital, Continent) :- capital(Capital, Country), continent(Country, Continent).:** Connects a capital with a continent

2. Set of query questions

To organize the data efficiently, we have used facts and predicates to create a knowledge base which can answer a variety of questions related to countries, regions, continents, and cities. With this knowledge base built using Prolog, you can ask a wide range of questions related to geography, neighbors, capitals:

1. What countries are neighboring Vietnam?
2. What is the capital of France?
3. Which countries are in Europe?
4. Which cities are in Southeast Asia?
5. What region is Egypt in?

6. Which country is neighboring Japan?
7. What is the capital of Poland?
8. Which country has the city of Lima as its capital?
9. Which countries are in Central America?
10. Which city is the capital of Egypt?
11. Which countries is neighboring Russia?
12. Which countries are in Asia?
13. Which countries have Moscow as their capital?
14. Which country is neighboring both Germany and Poland?
15. Which countries are in North America?
16. Which city is the capital of Egypt?
17. Which country is neighboring both Laos and China?
18. Which country is neighboring Brazil?
19. Which cities are capitals in Europe?
20. What continent is Brazil in?

IV. Implement logic deductive system in the programming language (Python)

1. Reasoning on first-order logic

Building a logical inference program using the forward deductive method (forward chaining). The algorithm uses references from the book *Artificial Intelligence: A Modern Approach.*

1.1. Improvement

The Deductive Algorithm mentioned above can work but not really effective. Here are some of the improvements applied to speed up the algorithm:

1. In the t-th loop, we only ignore the rules in which there are no conditions

can be satisfied based on the new knowledge arising from the t − 1 loop.

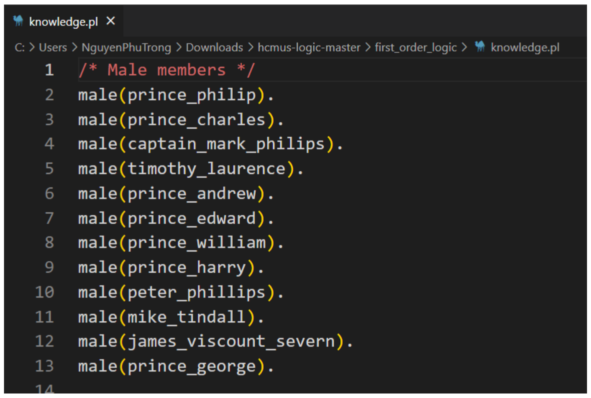
2. Assuming that conditional rules can only be joined by association, omit

through the step of normalizing the variables in the rule (Standardize-Variable(rule)).

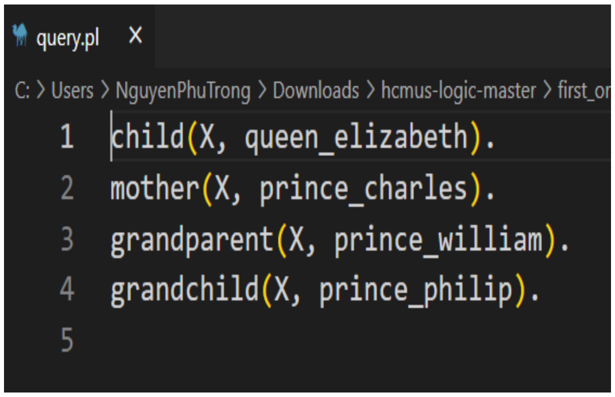
3. Extract a set of sentences that are likely to satisfy the condition of the law under consideration when generating combinations of pairs of sentences and find the suitable transformation.

1.2. Program description

● Input: a file containing the knowledge base and a file containing the user's questions.

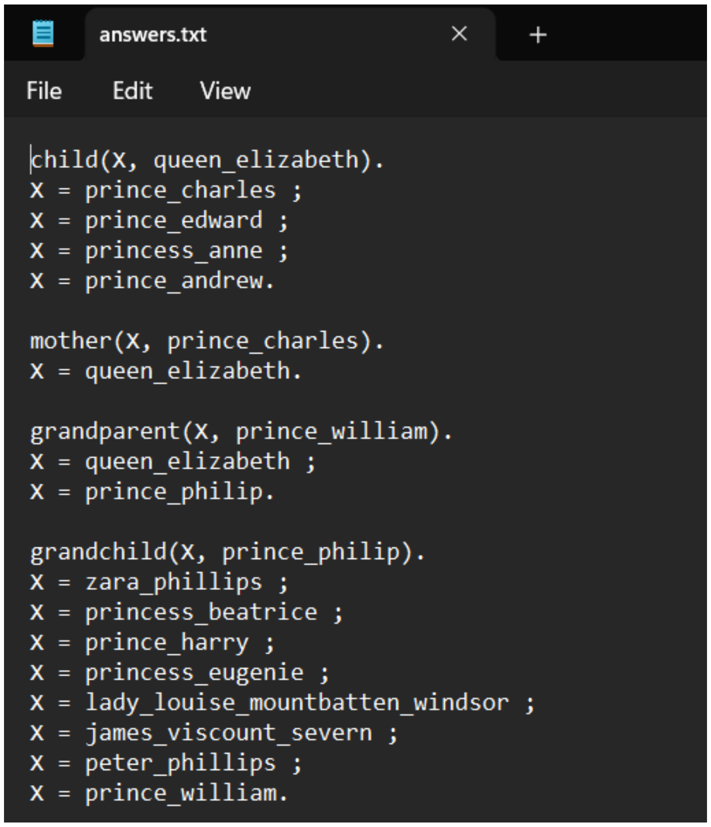


```
knowledge.pl ×
C: > Users > NguyenPhuTrong > Downloads > hcmus-logic-master > first_order_logic > knowledge.pl
/* Male members */
male(prince_philip).
male(prince_charles).
male(captain_mark_philips).
male(timothy_laurence).
male(prince_andrew).
male(prince_edward).
male(prince_william).
male(prince_harry).
male(peter_phillips).
male(mike_tindall).
male(james_viscount_severn).
male(prince_george).
```



```
query.pl ×
C: > Users > NguyenPhuTrong > Downloads > hcmus-logic-master > first_or
child(X, queen_elizabeth).
mother(X, prince_charles).
grandparent(X, prince_william).
grandchild(X, prince_philip).
```

● Output: the system's answer (inferred from the question based on existing knowledge). The output was written to "answers.txt".



```
answers.txt
File  Edit  View

child(X, queen_elizabeth).
X = prince_charles ;
X = prince_edward ;
X = princess_anne ;
X = prince_andrew.

mother(X, prince_charles).
X = queen_elizabeth.

grandparent(X, prince_william).
X = queen_elizabeth ;
X = prince_philip.

grandchild(X, prince_philip).
X = zara_phillips ;
X = princess_beatrice ;
X = prince_harry ;
X = princess_eugenie ;
X = lady_louise_mountbatten_windsor ;
X = james_viscount_severn ;
X = peter_phillips ;
X = prince_william.
```

● Predicate expression syntax is similar to that of Prolog, however:

○ Conditional rules can only be joined union operation.

○ Two-way relationship needs to be defined specifically, for example:

sibling(Person1, Person2) :- sibling(Person2, Person1).

Or friend(Person1, Person2) :- friend(Person2, Person1).

○ Queries have the same syntax as regular predicates, without the apostrophe single. For example, if you want to execute the query " What is the capital of France? ", we use the following syntax:

capical(X, France)

instead of: capital(X, “France”).

V. References

(1): https://www.swi-prolog.org/

(2): https://www.javatpoint.com/prolog

(3) https://github.com/HyrniT/logic

(4): hcmus-logic/first_order_logic at master · btcnhung1299/hcmus-logic (github.com)

(5): The book *Artificial Intelligence: A Modern Approach.*