

**Início** quarta-feira, 25 de dezembro de 2024 às 13:25**Estado** Prova submetida**Data de  
submissão:** quarta-feira, 25 de dezembro de 2024 às 15:25**Tempo gasto** 2 horas

## Informação

**Read these instructions carefully before you start the test.**

- This is an open-book test; you can use the materials available on the local computer and Moodle's course page, but no printed materials are allowed.
- You can use the resources on the computer, such as SICStus Prolog (and its manual).
- **The presence of electronic and/or communication devices is strictly forbidden.**
- The maximum duration of the test is stated below.
- The score of each question is stated in the test, totaling 20 points.
- A wrong answer in a multiple-choice question with four options implies a penalty equal to 25% of the question's value.
- A wrong answer in a True/False question implies a penalty equal to 50% of the question's value.
- Fraud attempts will be punished by the annulment of the test for all intervenients.
  
- **Use the predicate names and argument order exactly as specified in the test statement.**
- **Do not copy any code provided in the questions into the answer text box.**
- **You can use predicates requested in previous questions even if you haven't implemented them.**
- **However, do not include answers to previous questions in the current one.**
- **If you implement auxiliary predicates, you must include them in the answers to all questions where they are used.**
- **Pay attention to syntactic errors (such as forgetting the '!' after each rule/fact).**
- **Document your code and use intuitive variable names to clarify code interpretation.**

Be mindful of the time available.

Good work!

Consider the following knowledge base about movie sagas.

```
:-dynamic saga/4, movie/8.

%saga(SagaID, Saga Name, Number of Movies in Saga, Creator)
saga(1, 'Jurassic Park', 6, 'Michael Crichton').
saga(2, 'Indiana Jones', 4, 'George Lucas').
saga(3, 'Star Wars', 9, 'George Lucas').
saga(4, 'Harry Potter', 0, 'J. K. Rowling').
saga(6, 'Jaws', 0, 'Peter Benchley').

%movie(Movie Title, Year of Release, SagaID, Duration, IMDB Score, Director, Composer, Cast)
movie('Jurassic Park', 1993, 1, 127, 8.2, 'Steven Spielberg', 'John Williams', ['Sam Neill', 'Jeff Goldblum', 'Laura Dern', 'BD Wong']).
movie('The Lost World: Jurassic Park', 1997, 1, 129, 6.5, 'Steven Spielberg', 'John Williams', ['Jeff Goldblum', 'Julianne Moore', 'Vince Vaughn', 'Richard Schiff']).
movie('Jurassic Park III', 2001, 1, 92, 5.9, 'Joe Johnston', 'Don Davis', ['Sam Neill', 'William H. Macy', 'Téa Leoni']).
movie('Jurassic World', 2015, 1, 124, 6.9, 'Colin Trevorrow', 'Michael Giacchino', ['Chris Pratt', 'Bryce Dallas Howard', 'Irrfan Khan', 'BD Wong']).
movie('Jurassic World: Fallen Kingdom', 2018, 1, 128, 6.1, 'J.A. Bayona', 'Michael Giacchino', ['Chris Pratt', 'Bryce Dallas Howard', 'James Cromwell', 'BD Wong']).
movie('Jurassic World: Dominion', 2022, 1, 147, 5.6, 'Colin Trevorrow', 'Michael Giacchino', ['Chris Pratt', 'Bryce Dallas Howard', 'Campbell Scott', 'BD Wong']).

movie('Raiders of the Lost Ark', 1981, 2, 115, 8.4, 'Steven Spielberg', 'John Williams', ['Harrison Ford', 'Karen Allen', 'John Rhys-Davies']).
movie('The Temple of Doom', 1984, 2, 118, 7.5, 'Steven Spielberg', 'John Williams', ['Harrison Ford', 'Kate Capshaw', 'Ke Huy Quan']).
movie('The Last Crusade', 1989, 2, 127, 8.2, 'Steven Spielberg', 'John Williams', ['Harrison Ford', 'Alison Doody', 'Sean Connery']).
movie('Kingdom of the Crystal Skull', 2008, 2, 122, 6.2, 'Steven Spielberg', 'John Williams', ['Harrison Ford', 'Karen Allen', 'Shia LaBeouf']).

movie('The Phantom Menace', 1999, 3, 136, 6.5, 'George Lucas', 'John Williams', ['Ewan McGregor', 'Liam Neeson', 'Natalie Portman', 'Ian McDiarmid']).
movie('Attack of the Clones', 2002, 3, 142, 6.6, 'George Lucas', 'John Williams', ['Ewan McGregor', 'Hayden Christensen', 'Natalie Portman', 'Christopher Lee']).
movie('Revenge of the Sith', 2005, 3, 140, 7.6, 'George Lucas', 'John Williams', ['Ewan McGregor', 'Hayden Christensen', 'Natalie Portman', 'Christopher Lee']).
movie('A New Hope', 1977, 3, 121, 8.6, 'George Lucas', 'John Williams', ['Harrison Ford', 'Mark Hamill', 'Carrie Fisher', 'Alec Guinness']).
movie('The Empire Strikes Back', 1980, 3, 124, 8.7, 'Irvin Kershner', 'John Williams', ['Harrison Ford', 'Mark Hamill', 'Carrie Fisher', 'Billy Dee Williams']).
movie('Return of the Jedi', 1983, 3, 131, 8.3, 'Richard Marquand', 'John Williams', ['Harrison Ford', 'Mark Hamill', 'Carrie Fisher', 'Ian McDiarmid']).
movie('The Force Awakens', 2015, 3, 138, 7.8, 'J. J. Abrams', 'John Williams', ['Daisy Ridley', 'Harrison Ford', 'Mark Hamill', 'Carrie Fisher']).
movie('The Last Jedi', 2017, 3, 152, 6.9, 'Rian Johnson', 'John Williams', ['Daisy Ridley', 'Mark Hamill', 'Carrie Fisher', 'John Boyega']).
movie('The Rise of Skywalker', 2019, 3, 141, 6.4, 'J. J. Abrams', 'John Williams', ['Daisy Ridley', 'Mark Hamill', 'John Boyega', 'Adam Driver']).
```

Answer questions 1 to 5 **WITHOUT** using multiple solution predicates (findall, setof, and bagof), and **WITHOUT** using any SICStus library.

**Pergunta 1**

Pontuação 1,500

Implement *same\_saga(?Movie1, ?Movie2)*, which succeeds if the two (distinct) movies belong to the same saga, failing otherwise.



```
same_saga(Movie1, Movie2):-  
    movie(Movie1, _Y1, SagaID, _D1, _I1, _Di1, _C1, _Cs1),  
    movie(Movie2, _Y2, SagaID, _D2, _I2, _Di2, _C2, _Cs2),  
    Movie1 \= Movie2.  
  
% Notes:  
% - the verification that the movies are different should only be made after ensuring they are instantiated
```

**Pergunta 2**

Pontuação 1,500

Implement *movie\_from\_saga(?Movie, ?Saga)*, which relates a movie title with the name of the saga it belongs to.

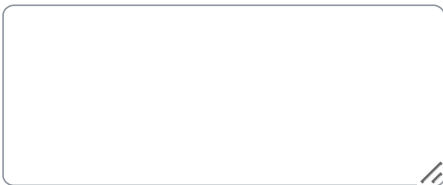


```
movie_from_saga(Movie, Saga):-  
    saga(SagaID, Saga, _N, _Creator),  
    movie(Movie, _Y, SagaID, _Dur, _I, _D, _C, _Cs).
```

**Pergunta 3**

Pontuação 1,500

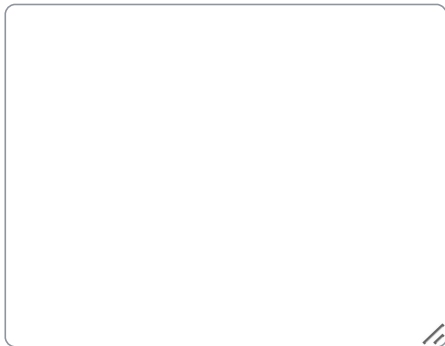
Implement *saga\_longest\_movie(?Saga, ?Movie)*, which relates a saga name with the movie title of the longest movie in the saga.



```
saga_longest_movie(Saga, Movie):-  
    saga(SagaID, Saga, _N, _Creator),  
    movie(Movie, _Y1, SagaID, Dur, _IM1, _Dir1, _Comp1, _Cast1),  
    \+(( movie(_M2, _Y2, SagaID, D2, _IM2, _Dir2, _Comp2, _Cast2), D2 > Dur )).  
  
% Notes:  
% - identical to exercise 8.a) from sheet P02 (we just have to deny the existence of a movie from the same saga with a longer duration)
```

Implement *add\_movie\_to\_saga(+Saga, +Movie, +Year, +Duration, +Score, +Director, +Composer, +Cast)*, which updates a given saga, adding a new movie to it with the provided information. If the specified movie already exists, the predicate should fail. Example:

```
| ?- add_movie_to_saga('Indiana Jones', 'The Dial of Destiny', 2023, 154, 6.7, 'James Mangold', 'John Williams', ['Harrison Ford', 'Karen Allen', 'John Rhys-Davies', 'Antonio Banderas']).
yes
| ?- add_movie_to_saga('Indiana Jones', 'The Dial of Destiny', 2023, 154, 6.7, 'James Mangold', 'John Williams', ['Harrison Ford', 'Karen Allen', 'John Rhys-Davies', 'Antonio Banderas']).
no
| ?- saga(_ID, 'Indiana Jones', NMovies, Creator).
NMovies = 5,
Creator = 'George Lucas'
```



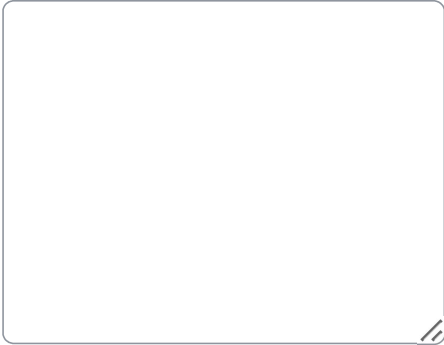
```
add_movie_to_saga(Saga, Movie, Year, Duration, Score, Director, Composer, Cast):-
    saga(SagaID, Saga, _N, _C),
    movie(Movie, Year, SagaID, Duration, Score, Director, Composer, Cast), !, fail.
add_movie_to_saga(Saga, Movie, Year, Duration, Score, Director, Composer, Cast):-
    retract( saga(SagaID, Saga, OldN, Creator) ),
    NewN is OldN + 1,
    assertz( saga(SagaID, Saga, NewN, Creator) ),
    assertz( movie(Movie, Year, SagaID, Duration, Score, Director, Composer, Cast) ).
```

% Notes:

% - If the movie already exists, no change should be made

% - In case it does not exist (and only if it does not exist), the movie should be inserted and the saga updated

Implement *movies\_from\_saga(+Saga, -Movies)*, which returns in *Movies* a list of all movie titles from *Saga*, ordered by year of release.



```

movies_from_saga(Saga, Movies):-
    saga(SagaID, Saga, _N, _C),
    movies_from_saga(SagaID, [], Movies).
movies_from_saga(SagaID, Movies, Final):-
    movie(Movie, Year, SagaID, _Dur, _Score, _Dir, _Comp, _Cast),
    \+ member(Year-Movie, Movies), !,
    movies_from_saga(SagaID, [Year-Movie|Movies], Final).
movies_from_saga(_SagaID, AllMovies, Movies):-
    sort(AllMovies, SortedMovies),
    remove_year(SortedMovies, Movies).

remove_year([], []).
remove_year([_Y-Movie | T], [Movie | T2]):-
    remove_year(T, T2).

% Notes:
% - this is a possible solution to obtain all movies (similar to the one presented in slide 3 from slides PL5)
% - another solution would be to use assert/retract (similar to the one presented in slide 17 from slides PL6)
% - it is necessary to order by year (hence the pairs Year-Movie), since there is no guarantee that movies are ordered in the
knowledge base
% - ordering can be achieved using either sort or keysort (built-in predicates presented in slide 54 from slides PL2)
% - note that the cut is necessary to avoid obtaining additional (incomplete) answers via backtracking

```

#### Informação

In the following questions, you can use multiple solution predicates (findall, setof, and bagof) as well as any SICStus library.

**Pergunta 6**

Pontuação 1,500

Implement *saga\_cast(+Saga, -Cast)*, which returns in *Cast* a list of all actors (in any order) who participated in movies from *Saga*. Avoid duplicate actors in the resulting list. Example:

```
| ?- saga_cast('Indiana Jones', Cast).  
Cast = ['Alison Doody','Harrison Ford','John Rhys-Davies','Karen Allen','Kate Capshaw','Ke Huy Quan','Sean Connery','Shia LaBeouf']
```



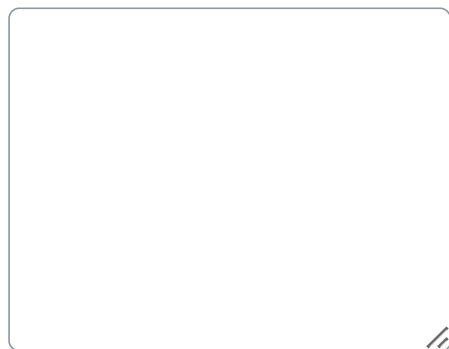
```
saga_cast(Saga, Cast):-  
    saga(SagaID, Saga, _N, _Cr),  
    findall(Actor, ( movie(_T, _Y, SagaID, _Dur, _Sc, _Dir, _Comp, Cast), member(Actor, Cast) ), List),  
    sort(List, Cast).  
  
% Notes:  
% - the sort/2 predicate orders a list, removing duplicates, thus obtaining the desired result (no duplicates)  
% - alternatively, the setof predicate, which includes that functionality, could be used  
% - in the case of using setof, it is necessary to use existential quantifiers for all remaining variables (see slides 10 and 11 from slides PL5)
```

**Pergunta 7**

Pontuação 1,500

Implement *sample\_cast(+Saga, -Cast)*, which returns a sample of the cast from *Saga*, by selecting only the actors in odd positions in the list of actors who participated in *Saga* (as given by the previous question). Example:

```
| ?- sample_cast('Indiana Jones', Cast).  
Cast = ['Alison Doody','John Rhys-Davies','Kate Capshaw','Sean Connery']
```



```
sample_cast(Saga, SampleCast):-  
    saga_cast(Saga, Cast),  
    sample(Cast, SampleCast).  
  
sample([], []).  
sample([E], [E]).  
sample([A,_B|T], [A|T2]):-  
    sample(T, T2).  
  
% Notes:  
% - it is necessary to take into account cases when the original list (as returned by saga_cast/2) has an even / odd number of elements  
% - a findall could be used to find all members of Cast in odd positions: findall(Actor, (nth1(Idx,Cast,Actor), Idx mod 2 =:= 1), SampleCast)
```

**Pergunta 8**

Pontuação 1,500

Implement *composer\_rating(+Composer, ?AvgScore)*, which unifies *AvgScore* with the average score of the movies for which *Composer* wrote the music.



```
composer_rating(Composer, AvgScore):-  
    findall(Score, movie(_M, _Y, _SID, _D, Score, _Dir, Composer, _Cs), Scores),  
    length(Scores, Len),  
    sumlist(Scores, Sum),  
    AvgScore is Sum / Len.
```

**Informação**

Robin wanted to implement a predicate *most\_successful\_composer(+Composer1, +Composer2, ?MostSuccessful)*, which unifies the third argument with the most successful of the two composers received in the first two arguments (the one with the highest average score). In the case of ties, the third argument can be unified with any of the first two.


He wrote the code below to solve this problem, and it seemed to work correctly until his colleague Bruce told him the code didn't work as intended.

```
most_successful_composer(Composer1, Composer2, Composer1):-  
    composer_rating(Composer1, R1),  
    composer_rating(Composer2, R2),  
    R1 >= R2, !.  
most_successful_composer(_Composer1, Composer2, Composer2).
```

**Pergunta 9**

Pontuação 0,500

Can you identify the problem with Robin's implementation?



When the third argument is instantiated and is equal to the second argument, the predicate always succeeds, even if that is not the most successful composer.

% Notes:

% - identical problem to the one presented in slide 12 from slides PL3

**Pergunta 10**

Pontuação 0,500

Is the cut in the code red or green? Justify your answer.

Red. If removed, Composer2 would always be a possible answer.

**Pergunta 11**

Pontuação 1,000

Re-implement the predicate so it works correctly in all situations.

```
most_successful_composer(Composer1, Composer2, Composer1):-  
    composer_rating(Composer1, S1),  
    composer_rating(Composer2, S2),  
    S1 >= S2.  
most_successful_composer(Composer1, Composer2, Composer2):-  
    composer_rating(Composer1, S1),  
    composer_rating(Composer2, S2),  
    S2 >= S1.
```

% Notes:

% - when both composers have identical ratings, both are possible results, via backtracking (non-discrimination)

**Informação**

We want to write facts in the form

Composer composed\_for Movie

Example:

'John Williams' composed\_for 'Jurassic Park'



**Pergunta 12**

Pontuação 0,500

Which is the best way to define *composed\_for* as an operator?

- ☐ a. `:-op(500, xfy, composed_for).`
- ☐ b. `:-op(500, xfx, composed_for).`
- ☐ c. `:-op(500, yfy, composed_for).`
- ☐ d. `:-op(500, fx, composed_for).`

Resposta correta:

`:-op(500, xfx, composed_for).`

**Pergunta 13**

Pontuação 1,000

Implement the necessary code so that code in this form can be used.

```
Composer composed_for Movie:-  
    movie(Movie, _Year, _Saga, _Dur, _Score, _Dir, Composer, _Cast).
```

% Notes:

% - similar to the example presented in slide 20 from slides PL7

**Informação**

Two people are considered directly connected if they worked in the same movie (as Director, Composer, or Actor).

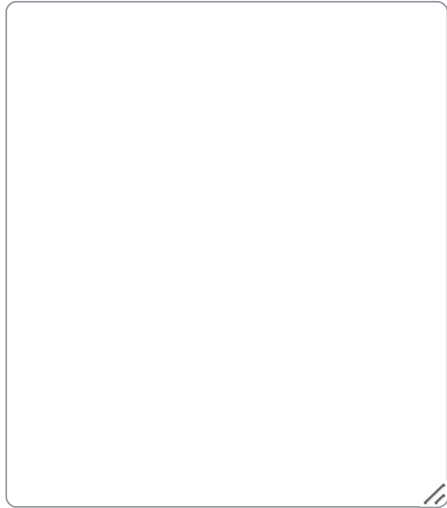
The following code implements the *connected(?Person1, ?Person2)* predicate, which succeeds if two people are directly connected.

```
connected(Person1, Person2):-  
    connected2(Person1, Person2),  
    Person1 \= Person2.  
connected(Person1, Person2):-  
    connected2(Person2, Person1),  
    Person1 \= Person2.  
  
connected2(Person1, Person2):-  
    movie(_T, _Y, _S, _D, _Sc, Person1, Person2, _).  
connected2(Person1, Person2):-  
    movie(_T, _Y, _S, _D, _Sc, Person1, _C, Cast),  
    member(Person2, Cast).  
connected2(Person1, Person2):-  
    movie(_T, _Y, _S, _D, _Sc, _Dir, Person1, Cast),  
    member(Person2, Cast).  
connected2(Person1, Person2):-  
    movie(_T, _Y, _S, _D, _Sc, _Dir, _Comp, Cast),  
    member(Person1, Cast),  
    member(Person2, Cast).
```

The Bacon number of an actor is the number of degrees of separation he or she has from Kevin Bacon. By extending the concept, we can determine the degree of separation between any two people involved in movies (Director, Composer, and Actors).

Implement *connected\_degree(+Person1, +Person2, ?Degree)*, which determines the degree of separation between two people - the minimum amount of 'connections' necessary to reach the other person. Examples:

```
| ?- connected_degree('Steven Spielberg', 'John Williams', Degree).
Degree = 1 ?
| ?- connected_degree('Don Davis', 'John Williams', Degree).
Degree = 2 ?
| ?- connected_degree('Don Davis', 'Harrison Ford', Degree).
Degree = 3 ?
```



```
connected_degree(Person1, Person2, Degree):-
    connected_degree_bfs([ [Person1] ], Person2, Degree).

connected_degree_bfs([ [Person1|R] | _], Person1, Degree):- !,
    length(R, Degree).
connected_degree_bfs([ [Person1|R] | T ], Person2, Degree):-
    setof(Next, ( connected(Person1, Next),
                  \+ (member(Next, [Person1|R])) ), List),
    append_all(List, [Person1|R], ToSee),
    append(T, ToSee, NextList),
    connected_degree_bfs(NextList, Person2, Degree).

append_all([], _List, []).
append_all([H|T], List, [ [H|List] |NT ]):-
    append_all(T, List, NT).

% Notes:
% - This is a breadth-first search, saving the path to determine its size
% - Explicitly said during the lecture (slide 18 from slides PL5) that this would be a good exercise for the test
% - A similar implementation is also requested in exercise 4 from sheet P05
```

Considering the expansion of the knowledge base to all known movie sagas and a Prolog interpreter that indexes clauses by all arguments, what would be the most efficient and correct implementation for the predicate *composed\_for\_saga(?Composer, ?Saga)?*

- ☐ a. 

```
composed_for_saga(Composer, Saga):-
    movie(_T, _Y, SagaID, _D,
        _Sc, _Dir, Composer, _),
    saga(SagaID, Saga, _N,
        _C).
```
- ☐ b. 

```
composed_for_saga(Composer, Saga):-
    nonvar(Saga),
    saga(SagaID, Saga, _N,
        _C),
    movie(_T, _Y, SagaID, _D,
        _Sc, _Dir, Composer, _).
composed_for_saga(Composer, Saga):-
    movie(_T, _Y, SagaID, _D,
        _Sc, _Dir, Composer, _),
    saga(SagaID, Saga, _N,
        _C).
```
- ☐ c. 

```
composed_for_saga(Composer, Saga):-
    nonvar(Composer), !,
    movie(_T, _Y, SagaID, _D,
        _Sc, _Dir, Composer, _),
    saga(SagaID, Saga, _N,
        _C).
composed_for_saga(Composer, Saga):-
    saga(SagaID, Saga, _N,
        _C),
    movie(_T, _Y, SagaID, _D,
        _Sc, _Dir, Composer, _).
```
- ☐ d. 

```
composed_for_saga(Composer, Saga):-
    saga(SagaID, Saga, _N,
        _C),
    movie(_T, _Y, SagaID, _D,
        _Sc, _Dir, Composer, _).
```

% Notes:

% - For efficiency, search should always start with the instantiated arguments (as seen in slide 5 from slides PL7)

% - Note that the double definition requires a cut to prevent solution duplication

% - Due to a lapse in the code (non\_var vs nonvar), no penalties will be applied to this question

Resposta correta:

```
composed_for_saga(Composer, Saga):-
    nonvar(Composer), !,
    movie(_T, _Y, SagaID, _D, _Sc, _Dir, Composer, _),
    saga(SagaID, Saga, _N, _C).
composed_for_saga(Composer, Saga):-
    saga(SagaID, Saga, _N, _C),
    movie(_T, _Y, SagaID, _D, _Sc, _Dir, Composer, _).
```

Considering Prolog code consisting only of pure logic predicates (i.e., without cuts, arithmetic operations, or other non-logical features), what would be the effect of reversing the order of the goals in each clause when executing a query?

- ☐ a. The same results would be obtained in the same order.
- ☐ b. Different results would be obtained.
- ☐ c. The same results would be obtained but in a different order.
- ☐ d. The code would become invalid, and it would no longer work.

Resposta correta: The same results would be obtained but in a different order.

Consider the following code and the existence of a *predX/2* predicate:

```
foreach(Elem1, List1), foreach(Elem2, List2) do predX(Elem1, Elem2).
```

Which call to a predicate from the *lists* library is it equivalent to?

- ☐ a. `scanlist(predX, List1, List2, Elem1, Elem2).`
- ☐ b. `select(Elem1, List1, Elem2, List2).`
- ☐ c. `delete(List1, Elem1, Elem2, List2).`
- ☐ d. `maplist(predX, List1, List2).`

% Notes:

% - similar example presented in slide 21 from slides PL8 (which we saw at the time to be equivalent to `maplist`)

Resposta correta:

```
maplist(predX, List1, List2).
```