CPS721: Assignment 1

Due: September 23, 2025, 9pm
Total Marks: 120 (worth 4% of course mark)
In groups of no more than 3

Late Policy: Late submissions are penalized by $3^n\%$ when they are n days late.

Clarifications and Questions: Please use the discussion forum on the D2L site to ask questions. These will be monitored regularly. A Frequently Asked Questions (FAQ) page will also be created. Only ask questions by email if it is specific to your submission. Any questions asked with less than 12 hours left to the deadline are not likely to be answered.

PROLOG Instructions. Your code must run on ECLiPSe 7.1. If you cannot install this on your own machine, test it on the lab machines or the department moon servers.

You are NOT allowed to use ";" (disjunction), "!" (cut), "->" (if-then), `setof`, `bagof`, `findall`, or any other language features and library predicates that are not covered in class. Similarly, you should only use "=" and "is" for equality, and "not" for negation. Do NOT use "=:=", "\=", or other such commands. You will be penalized if you use these commands/features. You are only allowed to use ";" to get additional responses when interacting with PROLOG from the command line (this is equivalent to using the "More" button in the ECLiPSe GUI).

Academic Integrity and Collaboration Policy: Any submission you make must be the work of your group and your group alone. You are also not allowed to post course materials (including assignments, slides, etc.) anywhere on the web, and you are also prohibited from using generative AI systems or online "help" websites for completing assignments.

Submission Instructions: You should submit ONE zip file called `assignment1.zip` on D2L. It should contain the following files:

```
assignment1_submission_info.txt
q1_movie_kb.pl        q1_queries.pl        q1_log.txt
q2_grocery_kb.pl      q2_rules.pl          q2_log.txt
q3_bacon1.pl          q3_bacon2.pl         q3_bacon3.pl          q3_log.txt
```

All these files have been given to you and you should fill them out using the format described. Ensure your names appear in all files — especially `a1_submission_info.txt` — and your answers appear in the file sections as requested. Do NOT edit, delete, or add any lines that begin with "`%%%%% SECTION:`". These are used by the auto-grader to locate the relevant code. You will lose marks if you edit such lines.

Your submission should not include any other files than those above. You will be penalized for using compression formats other than `.zip`. Submissions by email will not be accepted.

You may submit your solution multiple times. Doing so will simply overwrite your previous submission. We will therefore always mark the last submission your group makes.

1 A PROLOG Knowledge Base [40 marks]

For this problem, you will be making a knowledge base (KB) about movies and then you will write queries to answer questions about it.

a. [**10 marks**] Create a knowledge base by adding atomic propositions to the file `q1_movies_kb.pl`. The KB should ONLY contain atomic statements that use the following predicates:

- `releaseInfo(MovieName, Year, Length)` - a movie named `MovieName` was released in year `Year` and is `Length` minutes long.
- `directedBy(MovieName, Director)` - the movie named `MovieName` was directed by `DirectorName`.
- `actedIn(Name, Movie, CharacterName)` - an actor with name `Name` plays a character named `CharacterName` in the move named `Movie`.

Your knowledge base should include at least 10 different movies. Each movie should have a single release year, and each movie should have at least one director and one actor.

The purpose of this part of the assignment is to get you practice building a knowledge base. The information in the KB may be entirely made up, but you will also be using it for testing your queries in part (b). You may therefore find it useful if you choose atomic statements that allow you to test your queries effectively. You may also reuse the KB for question 3 — though you may also create an entirely new KB for that purpose — so consider how the statements may be useful for that purpose as well.

Please ensure you also adhere to the following:

- See file `q1_movies_kb.pl` for formatting instructions.
- You can assume that no two movies in the KB share the same title and that no two directors have the same name.
- For any movie, there is exactly one atomic statement using the `movieInfo` predicate (*ie.* each movie has a single release year and length).
- Movies may have more than one director and multiple actors.
- Remember that all movie, person, or character names should be in lower case. Replace any spaces in names or locations with underscores. For example, you may have a statement that `the_godfather` was directed by `francis_ford_coppola`. For simplicity, do not include any accents in the names. Do NOT use strings like "Francis Ford Coppola".

b. [**25 marks**] Create queries for each of the 11 statements below and add them to the file `q1b.pl`.

1. (1 mark) Did Steven Spielberg direct "Jaws"?
2. (1 mark) What character did Janelle Monáe play in "Glass Onion"?
3. (2 marks) Which actor was in both "Oppenheimer" and "Midsommar"?
4. (2 mark) Did any actor who appeared in "The Godfather", not appear in "The Godfather Part II"?
5. (2 mark) Did anyone direct a movie before 1980 and after 2010?
6. (2 marks) Did any actor appear in more than one movie in 2023?
7. (3 marks) Did any actor appear in more than one movie in the same year that were all at least 3 hours long?
8. (3 marks) Is there a year from 2010 to 2019 (inclusive) in which Sarah Polley did not direct a movie?

9. (3 marks) Did anyone act in 3 movies that were released in 3 consecutive years?
10. (3 marks) What is the oldest movie in the knowledge base?
11. (3 marks) What is the longest movie that Cate Blanchett has acted in?

Please note the following before completing this question:

- Ensure that you follow the instructions for formatting as specified in `q1_queries.pl`.
- You can then test your queries by compiling this file. It is already set to import the KB file in part (a), so you don't need to first compile that file as well.
- Your queries should capture the logic of the given statement, not just an answer specific to your knowledge base. For example, we will test your queries on different knowledge bases to make sure they do not only work in your KB.
- It is ok if your queries return the same result multiple times when you ask for more solutions. However, they should not return incorrect solutions.
- You can only use the predicates listed above with variables or constants as arguments, along with conjunction (","), equality ("="), and "not" (*ie.* negation) in your queries. You may also use $<, >, =<$, or $>=$, which PROLOG uses for less than, more than, etc. Ensure you also follow the Prolog instructions listed on page 1.
- Recall that when using the predicate `X` $<$ `Y`, both `X` and `Y` must be instantiated before the comparison. Similarly, be careful when using "not" on an expression that includes a variable.
- You will notice that facts for `year` have already been provided to you in `q1_queries.pl`. These facts may be useful for some of the queries. Do NOT edit this section of the file.

c. [5 marks] Test your queries from part (b) on your KB, and log the output. You should put the log of this interaction in the `q1_log.txt` file. This can simply be a copy paste of the 11 queries and their answer. Please ensure the difference between the queries and the answers are clear, and use extra space to separate one query from the next.

For at least 2 of the queries, you should also call "More" (or ';' in the terminal) to find all answers to the given query. Include the results of all queries even if you don't successfully complete all of them.

2 Arithmetic in PROLOG [30 marks]

This question will give you practice doing numerical calculations in PROLOG using the example of calculating a grocery bill.

a. [**10 marks**] You will be calculating the bill at a grocery store on five products: `milk`, `tomato`, `orange`,`marshmallow`, and `ice_cream`. For this step, you should create a knowledge base that contains atomic statements that use the following predicates:

- `cost(Product, Cost)` - defines the standard cost of each unit of a product.
- `numPurchased(Product, Count)` - indicates the number of the corresponding product that have been purchased. `Count` must be a non-negative integer.
- `twoForOneSale(Product)` - indicates that the product is on a two-for-one sale. Note that this means that someone purchases 5 of a product that is on a two-for-one sale, then the price will be equivalent to buying 3 of the product.
- `taxable(Product)` - indicates that the product is taxable. If a product does not have such a statement, it means tax is not applicable to it.
- `taxRate(Rate)` - the tax rate.

There should be exactly one `cost` and `numPurchased` statement for each of the five products. You may set the `Cost` and `Count` to any valid values of your choosing. At least one item should be taxable and at least one should be on a two-for-one sale. If you are using this KB for testing, it is worth picking statements that allow you to test a variety of cases. Finally, set the tax rate to anything of your choosing provided it is greater than 0 but less than 1 (*ie.* the tax rate of 13% would be 0.13).

All atomic propositions to `q2_kb.pl`. See that file for format instructions.

b. [**15 marks**] Now add rules that accomplish the following:

- Add rule `costPerUnitAfterTax(Product, AfterTax)` which calculates the price per unit of the given `Product` after tax has been added. If the item is not taxable, then the after tax value is the same as the original cost. If it is taxable, then the after tax value is given by the original value increased by the tax rate. For example, if the tax rate is 13% and the cost of a product is $1.00, then `AfterTax` should be set to 1.13.
- Add a rule, `costPerUnitAfterTaxAndSale(Item, AfterSaleAndTax)` that calculates the total for the number purchased of the given item after the sale is included. The value should include the tax if the item is taxable. It may be useful to use the "X mod Y" operator and the "X // Y" operator (integer division) in PROLOG for this purpose.
- Add a rule `totalCost(Cost)` that calculates the total cost for the five items purchased.

Include these rules in `q2_grocery_rules.pl`. Make sure to follow the formatting instructions in that file. In addition, note the following:

- You are allowed to have the different rules call each other, or to have more than one sentence per rule.
- You are allowed to add your own "helper" predicates.
- We will test your rules with other KBs, not just yours. Thus, ensure your rules do not hardcode the tax rate, the cost, etc. of any of the products. You should also not hardcode any product names (*ie.* `milk`, `tomato`, etc.), except in your definition of `totalCost`.
- Your rules should not return more than one answer when if you ask for "More".

- Remember to read the instructions on page 1 about the Prolog operators that you are not allowed to use.
- You can then test your queries by compiling `q2_grocery_rules.pl`. It is already set to import the KB file in part (a), so you don't need to first compile that file as well.

c. [**5 marks**] Create 5 total queries that test the `costAfterTax`, `costAfterTaxAndSale`, and `totalCost` rules. Any queries of your choosing on these three rules is allowed, as long as each is tested at least once. You should also add a comment above each query stating what the query checks in plain English.

The queries should be written in the file `q2_log.txt`. A log interaction when testing these queries should also be included. See the file for formatting instructions. Include the queries and interaction even if you don't successfully complete all rules.

3 Recursive Rules [50 marks]

"Six Degrees of Kevin Bacon" is a game where participants try to find a "path" of actors from a given actor to the actor Kevin Bacon. Here, the path consists of actors who appeared together in the same movie. For example, John Lithgow was in "Footloose" with Kevin Bacon, so John Lithgow can get to Kevin Bacon along a path of length one. Mike Myers was in "Shrek" with John Lithgow, so Mike Myers can get to Kevin Bacon along a path of length two using the path of Mike Myers to John Lithgow through "Shrek", and John Lithgow to Kevin Bacon through "Footloose".

For this question, you will implement programs for different variants of this game. You can test each by compiling the corresponding file for that variant. Doing so will import the KB from question 1 directly. Note, that you may add your own helper predicates. You are NOT allowed to use Prolog lists for any of the problems below. In addition, make sure you are familiar with the Prolog restrictions used in this class as outlined on page 1.

a. [**15 marks**] In the first version you will implement the standard version of this game with a predicate `canReach(A1, A2, M)`, which holds when there is a path of no more than `M` between actors `A1` and `A2`. Thus, your program will not necessarily find the shortest path, but it will hold if a path of no larger than the maximum distance `M` exists.

Your program should satisfy the following conditions:

- If `canReach(A1, A2, M)` holds, then `canReach(A1, A2, M2)` should hold for any `M2 > M`.
- The distance between an actor and their self is 0, and so `canReach(A, A, M)` should hold for any `M` ≥ 0, provided that actor `A` appears in some movie in the KB.
- `canReach` will always be input as an integer (*ie.* `canReach` will never be called with `M` being set as a variable).
- Your program should be able to handle standard cases like `canReach(kevin_bacon, john_lithgow, 2)`, where none of the inputs are variables.
- Your program is not required to avoid repeating actors or movies in the same path. Preventing some repetition may speed your program up, but is not necessary to do so.
- Your program should handle cases like `canReach(samuel_jackson, X, 2)` and `canReach(X, samuel_jackson, 2)`, which will find an actor who is at most two steps away from Samuel Jackson. This query should find all actors who can be reached in at most 2 steps from Samuel Jackson if "More" is repeatedly called.
- Your program should handle cases like `canReach(X, Y, 2)`, which will find all pairs of actors who can be reached from each other in at most 2 steps if "More" is repeatedly called.

The majority of the marks will be given for satisfying the first 4 conditions. Only extend your solution to handle the last two cases once you have the first 4 working.

Your rules should be put in `q3_bacon1.pl`. See that file for formatting instructions.

b. [**15 marks**] A more challenging version of "Six Degrees of Kevin Bacon" involves requiring that the path between the actors goes "through" a specific given movie, at least once. For example, consider the task of finding a path from Heather Graham to Kevin Bacon through the movie "Shrek". A path with 3 such steps exists, since Heather Graham was in "Austin Powers" with Mike Myers, who was in "Shrek" with John Lithgow, who was in "Footloose" with Kevin Bacon. However, this path does not prove that a path of length 3 exists between Heather Graham and Kevin Bacon that includes the movie "Cliffhanger", even though John Lithgow appeared in "Cliffhanger", because it was not included as part of a transition along the path.

To this end, you should write a predicate called `canReachThroughMovie(A1, A2, Movie, M)`, which holds if and only if there is a path in the KB from actor `A1` to `A2` such that the path includes `Movie` and has a length of at most `M`.

Consider the following when writing your program:

- A "self-loop" cannot be used to satisfy the movie requirement. For example, one cannot get from Mike Myers to Kevin Bacon through "Cliffhanger" with a path like Mike Myers to John Lithgow through "Shrek", John Lithgow to John Lithgow through "Cliffhanger", and John Lithgow to Kevin Bacon through Footloose.
- Because of the previous requirement, it is no longer the case that `canReachThroughMovie(A, A, Movie, 0)` holds, even if actor `A` was in `Movie`.
- Repetition is still allowed. As such, `canReachThroughMovie(john_lithgow, john_lithgow, shrek, 2)` holds due to the path of John Lithgow to Mike Myers through "Shrek", and Mike Myers to John Lithgow through "Shrek".
- As in part (a), you can always assume that `MaxDist` is given as a non-negative integer (*ie.* it is never given as a variable).
- As in part (a), most of the marks will be assigned for handling fully instantiated cases like `canReachThroughMovie(heather_graham, kevin_bacon, shrek, 3)`.
- For full marks, your solution should handle any other combination of variables, including `canReachThroughMovie(zendaya, X, dune, 3)`, `canReachThroughMovie(zendaya, X, M, 3)`, and `canReachThroughMovie(A1, A2, M, 3)`.

Your rules (and any additional helper predicates) should be put in `q3_bacon2.pl`. See that file for formatting instructions. If you want to reuse part of your solution to part (a), copy that code over to `q3_bacon2.pl` as detailed in the file.

c. [10 marks] We will now extend this game once more and require that the path goes through 2 given movies. To this end, you should write a predicate called `canReachThrough2Movies(A1, A2, Mov1, Mov2, M)`, which holds if and only if there is a path in the KB from actor `A1` to `A2`, such that the path includes both movies `Mov1` and `Mov2`, and has a length of at most `M`.

Consider the following when writing your program:

- Your program should allow `Mov1` and `Mov2` to be reached in any order. For example, the example path in part (b) means that `canReachThrough2Movies(heather_graham, shrek, footloose, 3)` and `canReachThrough2Movies(heather_graham, footloose, shrek, 3)` both hold.
- If `Mov1 = Mov2`, then at least two steps along the path must go through movie `Mov1`.
- As in part (a) and (b), you can always assume that `M` is input as an integer and is never given as a variable.
- Your program should be able to handle the combination of variables for `A1`, `A2`, `Mov1`, and `Mov2` may be variables for full marks.
- As in part (b), self-loops are not allowed for satisfying the movie requirements.

Your rules (along with any helper predicates) should be put in `q3_bacon3.pl`. See that file for formatting instructions. If you want to reuse parts of your solution to part (a) pr (b), ensure to copy that code over to `q3_bacon3.pl` as detailed in that file.

d. [10 marks] Create 5 queries for each of parts (a), (b), and (c), that test your rules on the KB from question 1. Enter these queries, along with an English language explanation of each, in the file `q3_log.txt`. You may include more queries if you choose, just ensure the same format is followed.

In addition, put a log of your tests at the end of this file. Include the queries and interaction even if you don't successfully complete all rules.