CPS721: Assignment 1

Due: September 24, 2024, 9pm
Total Marks: 100 (worth 4% of course mark)
You MUST work in groups of 2 or 3

Late Policy: The penalty for submitting even one minute late is 10%. Assignments are not accepted more than 24 hours late.

Clarifications and Questions: Please use the discussion forum on the D2L site to ask questions as they come up. These will be monitored regularly. Clarifications will be made there as needed. A Frequently Asked Questions (FAQ) page will also be created. You may also email your questions to your instructor, but please check the D2L forum and FAQ first.

PROLOG Instructions: When you write your rules in PROLOG, you are NOT allowed to use ";" (disjunction), "!" (cut), and "->" (if-then). You are only allowed to use ";" to get additional responses when interacting with PROLOG from the command line. Note that this is equivalent to using the "More" button in the ECLiPSe GUI.

We will be using ECLiPSE Prolog release 6 to mark the assignments. If you run any other version of PROLOG, it is your responsibility to check that it also runs in ECLiPSE Prolog release 6.

Academic Integrity and Collaboration Policy: You can only discuss this assignment with your group partners or with your CPS721 instructor. Recall that you are not allowed to post course materials (including assignments, slides, etc.) anywhere on the web. In addition, you are prohibited from using generative AI systems or online "help" websites for completing assignments. By submitting this assignment, you acknowledge that you have read and understood the course policy on collaboration and contract cheating as stated in the CPS721 course management form.

Submission Instructions: You should submit ONE zip file called `assignment1.zip`. It should contain 6 files:

`q1_bank_kb.pl`	`q1_queries.txt`
`q2_bank_calc.pl`	`q2_queries.txt`
`q3_robocup.pl`	`q3_queries.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, and your answers appear in the file sections as requested. Do NOT edit or delete any lines that begin with the "`%%%%% 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. If you submit a `.rar`, `.tar`, `.7zip`, or other compression format aside from `.zip`, you will lose marks. All submissions should be made on D2L. Submissions by email will not be accepted.

As long as you submit your assignment under the name `assignment1.zip` your group will be able to submit multiple times as it will overwrite an earlier submission. You do not have to inform anyone if you do. The time stamp of the last submission will be used to determine the submission time. Do not submit multiple `zip` files with different names. If you do, we will use the last submitted one, but you may lose marks.

If you write your code on a Windows machine, make sure the files are saved as plain text and are readable on Linux machines. Ensure your PROLOG code does not contain any extra binary symbols and that they can be compiled by ECLiPSE Prolog release 6. You can test this by ssh-ing onto the department servers (*ie.* `moon`) and running ECLiPSE remotely from the command-line.

1 A PROLOG Knowledge Base [35 marks]

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

a. [10 marks] Create a knowledge base by adding atomic propositions to the file `q1_bank_kb.pl`. The KB should ONLY contain atomic propositions, and they should ONLY use the following predicates:

- `hasAccount(Name, Bank, Balance)` - a person with name `Name` has an account at a bank with the name `Bank`. The amount in this account is `Balance`.
- `lives(Name, City)` - a person with name `Name` lives in `City`.
- `created(Name, Bank, M, Y)` - the bank account of `Name` at `Bank` was created in the month `M` and year `Y`. Here, `M` is an integer from 1 (January) to 12 (December).

You should write a knowledge base with at least 10 different people, each of which has at least 1 account. The purpose of this part of the assignment is to get you practice building a knowledge base. You will also be using it for testing your queries in part (b). You should therefore ensure that the answer to all the queries below return `yes`, and that the statements in your KB allow you to test your queries effectively.

A few additional important notes:

- Ensure that you put your predicates in the section with header "`q1_kb`". You will lose marks if you put the statements anywhere else.
- You can assume that a person only has one account at a bank at any time, but they may have accounts at different banks.
- For each person in the knowledge base, there should be exactly one `lives` fact about them (*ie.* no one lives in more than 1 city, and the city everyone lives in is known).
- For every `hasAccount` fact, there should also be exactly one `created` fact (*ie.* every account has exactly one creation date, and it is in the knowledge base).
- Remember that all person names, bank names, and city names should be in lower case. Replace any spaces in names or locations with underscores. For example, you may have a statement that `al_pacino` lives in `moose_jaw` and has an account at `cibc`. Do not include any accents in the names.
- You will notice that facts for `month` have already been provided to you. These will help with part b. DO NOT EDIT THIS SECTION.

b. [20 marks] Create queries for each of the 11 statements below and add them to the file `q1_queries.txt`. Please see the notes below for important things you should keep in mind while completing this question.

1. (1 mark) Does Ada Lovelace live in San Francisco?
2. (1 mark) What is the balance of Alan Turing's account with CIBC?
3. (1 mark) Does anyone have an account at both BMO and Scotiabank, and if so, who is it?
4. (1 mark) Is there a person with an account at Wells Fargo that does not live in New York?
5. (1 mark) Does anyone who opened an account with CIBC in April 2018 have more than $5000 in their CIBC account? In your interaction log (see part (c)), list all such people using the ";" command if you are using the command line or the "More" button if you are using the GUI.
6. (2 marks) Does any bank have more than one person who opened an account in January 2024?

7. (2 marks) Did any person living outside of Winnipeg open more than one account in the same year? In your interaction log (see part (c)), list all such people using the ";" command if you are using the command line or the "More" button if you are using the GUI.
8. (2 marks) Is there a month in 2017 in which no one opened an account at the National Bank of Canada? In your interaction log (see part (c)), list all such people using the ";" command if you are using the command line or the "More" button if you are using the GUI.
9. (3 marks) Did anyone open 3 different accounts in 3 consecutive years?
10. (3 marks) Who opened the oldest account ever opened at BMO?
11. (3 marks) Who has the largest account (*ie.* highest balance) created in 2010 or later, of anyone who lives in Toronto?

Please note the following before completing this question:

- Your queries should capture the logic of the given statement, not just an answer specific to your knowledge base. For example, if we changed the knowledge base, your queries should still return the correct answers for the new knowledge base.
- It is ok if your queries return the same result multiple times when you ask for more solutions. However, it should not return incorrect solutions.
- Ensure that you follow the format in `q1_queries.txt`. You must put the query for each part under the corresponding comment, and will lose marks if you do otherwise.
- You can only use the predicates listed above with variables or constants as arguments, conjunction, and "not" (*ie.* negation) in your queries. You may also use $<, >, =<$, or $>=$, which PROLOG uses for less than, more than, etc. In addition, make sure you read the instructions on page 1 which lists Prolog operators which are not allowed.
- Recall that when using the predicate $X < Y$, both X and Y must be instantiated before the comparison.
- To help with some of the queries, facts for the `month(M)` predicate has already been provided to you. These list the applicable months (as integers).
- An account opened in May 2024 was opened after an account opened in April 2024, which was opened after an account opened in December 2023. Keep this in mind when formulating queries involving terms like "older".

c. [**5 marks**] Test your queries from part (b) on your KB, and log the output. You should put the log of this interaction at the end of the `q1_queries.txt` file. Make sure the answers you get are for the final version of your KB. You will lose marks if our tests with your queries produce different answers than you list in this file.

2 Arithmetic in PROLOG [30 marks]

This question involves numerical calculations in PROLOG to extend the example given in question 1. In particular, you will calculate what the updated balance is for an account at the end of the month. This will take into account the interest rate offered by the account, the amount of deposits and withdrawals, the overdraft penalty, and the minimum needed to generate interest.

For example, suppose a person has $1,000 in an account at a bank with a monthly interest rate of 0.5%. Assume that the person deposits a total of $500 and withdraws a total of $250 from the account over the month. If the minimum account balance needed to generate interest is $1,000, then the interest accrued will be

$$(1,000 + 500 - 250) * (1 + 0.005) = 6.25$$

and the amount in the account at the end of the month will be 1256.25. However, if the minimum account balance needed to generate interest is 1,500, then no interest is accrued in the above example, and the balance at the end of the month is just $(1,000 + 500 - 250) = 1250$.

a. [**5 marks**] Create a knowledge base with information for at least 2 people (you can give them any names) that have bank accounts at each of two banks: `cibc` and `bmo`. The knowledge base should include facts for the following

- `totalDeposits(Name, Bank, Amount)` - the total deposits by `Name` at `Bank` over the last month is `Amount`.
- `totalWithdrawals(Name, Bank, Amount)` - the total withdrawals by `Name` at `Bank` over the last month is `Amount`.
- `monthlyRate(Bank, Rate)` - the monthly interest at bank `Bank` is `Rate`. Note that if the monthly rate is 0.5%, then the value if `Rate` should be 0.005.
- `interestLevel(Bank, MinLevel)` - the minimum amount at `Bank` that must be in the account after deposits and withdrawals in order to be eligible for interest.
- `penalty(Bank, PenaltyAmount)` - the penalty for having a balance below 0 at `Bank` is `PenaltyAmount`. For example, if the amount after the month is $−200 and the penalty is 50, then the total in the account is after the penalty is applied is $−250. Note that `PenaltyAmount` will always be a positive number.

Put the atomic propositions defining your knowledge base in the file `q2_bank_calc.pl` in the section labelled `q2_kb`. The initial account balance at the end of each month should be defined using `hasAccount` as defined in question 1. Notice that you will need two statements for each of `monthlyRate`, `penalty`, and `interestLevel` (one for each bank). For each person in your KB, you will also have 2 statements for each of `hasAccount`, `totalDeposits`, and `totalWithdrawals` (one for each bank). You may reuse statements from question 1, but these must be copied over into `q2_bank_calc.pl`.

Note, if you do not put the propositions in the `q2_kb` section of `q2_bank_calc.pl`, you will lose marks.

b. [**15 marks**] Add rules to `q2_bank_calc.pl` that accomplish the following:

- Add the rule `subtotal(Name, Bank, Subtotal)`, which calculates the amount in the account after including deposits and withdrawals, but not the accrued interest and overdraft penalty.
- Add a rule `accruedInterest(Name, Bank, I)` which calculates the accrued interest `I` at the end of the month for the person's account at `Bank`. In the example above, `Accrued` should be 6.25 if the minimum interest level is $1,000, and 0 if the minimum interest level is $1,500.

- Add a rule `accruedPenalty(Name, Bank, P)` which calculates the penalty `P` that is accrued at the end of the month for the person's account at `Bank`. Specifically, it should be equal to the bank's penalty if the subtotal is negative, and 0 otherwise. Note that this means that `P` ≥ 0.
- Add a rule `endOfMonthBalance(Name, Bank, Balance)` which calculates the end of month balance for the person's account at the given bank. This will include the monthly deposits, withdrawals, interest, and penalty.
- Add a rule `endOfMonthBalance(Name, Balance)` which calculates the end of month balance for the person's account at ALL banks. This is the sum of the end of month balance over both banks.

Include these rules in the "`q2_rules`" section of `q2_bank_calc.pl`. You will lose marks if they are not in the correct section. In addition, note the following:

- You are allowed to have the different rules call each other, or have more than one sentence per rule.
- You are allowed to add your own "helper" predicates. If you do (it is not required), you should also put these in the "`q2_rules`" section.
- We will test your rules with other KBs, not just your own. However, you can assume that all KBs will only refer to the two given banks, `cibc` and `bmo`, for each of which there will be statements defining each of the interest rate, penalty, and minimum interest level. In addition, for every person in the KB, there will be `hasAccount`, `totalDeposits`, and `totalWithdrawal` statements for each of the banks (*ie.* none of these statements will be missing).
- We may test with any of the parameters set as variables. For example, `subtotal(tom, Bank, X)` and `accruedInterest(P, cibc, X)` are valid queries.
- When testing your rules, it is ok if they return the same answer multiple times, but they should not return incorrect answers.
- Remember to read the instructions on page 1 about the Prolog operators that you are not allowed to use.

c. [**10 marks**] Create at least 10 queries that test the `subtotal`, `accruedInterest`, `accruedPenalty`, and the `endOfMonthBalance` rules. You should not use the ";" command in these queries except to generate additional answers. See the first page for additional restrictions on what operators you are allowed to use. Any queries of your choosing on these 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_queries.txt`. Make sure the answers you get are for the final version of your KB. You will lose marks if our tests with your queries produce different answers than you list in this file. In addition, put a log of your tests with these queries at the bottom of the `q2_queries.txt` file. You will lose marks for not doing so.

3 Recursive Rules [35 marks]

This problem will exercise what you have learned about defining recursive rules in PROLOG to answer questions about a knowledge base.

a. [**25 marks**] *RoboCup* (`www.robocup.org`) is an international joint project to promote AI, robotics, and related research. One of the grand challenges in this project is to have teams of robots play soccer. The robots participating in this challenge may need to reason to choose the most successful behavior.

In this part of the assignment, you are asked to write a set of control rules that will allow a robot to reason whether they can score from their current situation. The knowledge base will include statements defining the current situation using the following predicates:

- `robot(R)` - means that `R` is a robot.
- `pathClear(L1, L2)` - means that there is a clear path (*ie.* no opponent player in the way) from `L1` to `L2`. Here, `L1` and `L2` can either be a robot or the goal, which is denoted by the constant `goal`.
- `hasBall(R)` - means that robot `R` has the ball in the current situation.

Note that `pathClear(R, L2)` means that if `R` has the ball, then they can kick it to `L2` without it being intercepted by the opponent. The same is true of `pathClear(L2, R)`, since `pathClear` is a symmetric predicate.

You will now write rules for the following predicates:

- `canPass(R1, R2, M)` - means that if the robot `R1` has the ball, they can get the ball to `R2` in at most `M` passes. That is, either `R1` can pass the ball directly to `R1` (and `M` ≥ 1), or the ball can be passed through a sequence of robots so that it arrives to `R2` in at most `M` passes.
- `canScore(R, M)` - means that the robot `R` can score using at most `M` kicks, where a kick is either a pass or a shot at the goal. That is, `R` can score if they have the ball and can shoot directly at the goal (and `M` ≥ 1), or because the ball can be passed to `R` in at most `M - 1` steps.

Note, that for simplicity, we are assuming that the opponents are not moving (*ie.* if a path remains clear before a pass, then it also remains clear after the pass. For example, consider the situation defined by the following knowledge base.

```
robot(r1). robot(r2). robot(r3).
robot(r4). robot(r5). robot(r6).
hasBall(r3).
pathClear(r1, goal).
pathClear(r2, r1).
pathClear(r3, r2).
pathClear(r3, goal).
pathClear(r3, r1).
pathClear(r3, r4).
pathClear(r4, goal).
pathClear(r1, r5).
pathClear(r5, r6).
```

Here, `canScore(r3, 1)` holds since `r3` has the ball, and the path is clear between `r3` and the `goal`. It also holds that `canPass(r3, r1, 1)`, since the path since the path is clear between `r3` and `r1`. In turn, this means that `canScore(r1, 2)` holds, since `r3` can get `r1` the ball in 1 step, and the path is clear between the `r1` and the goal. Since `r3` can get the ball to `r2` with one pass, who can then pass it to `r1`, it is also true that `canScore(r2, 3)`.

In addition, note the following:

- Your rules should be put in the `q3_rules` section of the supplied file

- Since `canScore(r2, 3)` holds, it is also the case that `canScore(r2, 4)`, `canScore(r2, 5)`, etc. hold. However, `canScore(r2, 2)` does not hold, since it takes 1 pass to get `r2` the ball, and `r2` can't then get it to the goal without at least 2 more kicks.
- `canPass(R1, R2, M)` does not require that `R1` has the ball, only that if they ever get the ball, they can get it to `R2` in at most `M` passes.
- A robot cannot pass the ball to the goal, so `canPass(R, goal, M)` should always fail for any robot `R` or `M`.
- The KB for the situation above is already given in the supplied file. We will test your rules in situations (ie. KBs) other than the one given above. We may use any robot names, but the constant `goal` will always mean the goal.
- We will not test your `canPass` or `canScore` rules with queries in which `M` is a variable. That is, we will not perform tests of the form `canPass(r1, r2, M)` or `canScore(r1, M)`.
- Any of the other parameters may be variables. In particular, we may have tests of the form `canPass(r1, r2, 5)`, `canPass(R, r1, 5)`, `canPass(r1, R, 5)`, and `canPass(R1, R2, 5)`. Similarly, we may test queries of the form `canScore(r1, 4)` or `canScore(R, 4)`.
- If multiple answers are requested using ";" or "More" it is ok if your rules return duplicate answers. However, they should never return incorrect answers.
- You can assume only one robot has the ball in the knowledge base.
- If a robot `r1` can pass to at least one other robot, then `canPass(r1, r1, M)` should succeed for any `M` ≥ 2. This is because of the symmetrical nature of `pathClear`.
- You are allowed to define and use helper predicates.
- You may edit the KB given if you find it helpful for testing. However, your interaction log (see part (c)) should be based on the given KB.

b. [10 marks] Create 10 queries that test your rules on the situation given by the KB from part (a). Enter these queries, along with an English language explanation of each, in the file `q3_queries.txt`.

In addition, put a log of your tests with these queries at the bottom of the `q3_queries.txt` file. Make sure the answers you get are for KB given in part (a). You will lose marks if our tests with your queries produce different answers on this KB than you list in this file.