CPS721: Assignment 4.

Due: November 12, 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 Page will also be created. You may also email your questions to your instructor. Check the D2L forum and frequently asked questions first.

Collaboration Policy: You can only discuss this assignment with your group partners or with your CPS721 instructor. You cannot use any external resources (including those which are posted on the Web) to complete this assignment. Failure to do this will have negative effect on your mark. However, you are allowed to interact with any of the publicly available LLMs to complete the bonus question. By submitting this assignment you acknowledge that you read and understood the course Policy on Collaboration in homework assignments stated in the CPS721 course management form.

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.

SUBMISSION INSTRUCTIONS: You should submit ONE zip file called `assignment4.zip` containing 3 files:

```
bankAccounts.pl    testQueries.txt        interaction.txt   [bonus.pdf]
```

These files have been given to you and you should follow the format given. Submitting your solution to the bonus question is optional. Your submission should not include any other files. 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 with the file name `assignment4.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.

Make sure the files are saved in plain text and 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.

This assignment will exercise what you have learned about natural language understanding. In this assignment, we imagine a computerized banking assistant. Ultimately, we would like to be able to tell our assistant what to do using ordinary English imperative sentences like "Transfer $500 from my Bank of Montreal account to the small account of my partner in the Metro Credit Union."[1] As in class, to interpret English sentences like these, we need to construct a *database* of facts about our world, a *lexicon*, and a *parser/interpreter*. As in class, we will restrict our attention here to the English noun phrases like "the large account of a woman from Scarborough in a Canadian bank", where "of" can be considered as a shorthand for "owned by".

1 Building a Database [10 marks]

Before we start using any English words, build in Prolog a simple database of facts (e.g., 10 facts of each type) about imaginary accounts, their owners and banks. To do so, add atomic propositions to the file `bankAccount.pl` in the section called `database`. The atomic propositions should ONLY use the following predicates:

- $account(AccountID, Name, Bank, Balance)$ where $AccountID$ is an account ID (a positive integer), $Name$ is a name of a person who holds this account, $Bank$ is a name of a bank where this account is located, and $Balance$ is the current amount stored in the account. For example:
```
account(12,ann,metro_credit_union,4505).
account(13,robert,royal_bank_of_canada,1091).
```
- $created(AccountID, Name, Bank, Month, Year)$ where $AccountID$ is an account ID as before, $Name$ is the name of the owner, $Month$, $Year$ are the month and the year when the account was opened. For example:
```
created(12,ann,metro_credit_union,8,2023).
created(13,robert,royal_bank_of_canada,6,2024).
```
- $lives(P, City)$, where P is a person name and $City$ is a city. For example:
```
lives(philip,richmondHill).  lives(ann,markham).
```
- $location(X, C)$, where either X is a city and C is a country, or X is bank and C is a city (include about 10 facts for both alternatives). The cities and banks should be same that you mentioned before. For example:
```
location(scarborough,canada).  location(markham,canada).
location(sanFrancisco,usa).  location(royal_bank_of_canada,toronto).
```
- $gender(Name, X)$ specifies that a gender of a person $Name$ is X, where X may take on the values `man`, `woman`, or other values, but we will only test the programs with `man`, `woman`.

When you develop your knowledge base of facts, consider the following:

- You will use this knowledge base for testing in the subsequent parts of this assignment, so we suggest you read the rest of the assignment first to see what kind of atomic facts will be useful to fully test your system.
- You will lose marks if your KB includes facts stated with predicates other than `account(), created(), lives(), location(), geneder()`.
- As on Assignment 1, use constants for your person, location and bank names. For simplicity, you should also not include any punctuation or accents in names as well.
- No bank will have more than one account with the same ID.
- You can assume that every person name, city name, and bank name is unique.

[1]Of course, executing these commands will require proper authorization and verification of access privileges to an account. This might be a topic for another course.

- Every person lives at exactly one location
- Unlike Assignment 1, a person may have multiple accounts at the same bank. Such accounts must have different IDs. A person may still have accounts at multiple banks as well.
- Not all individuals will have a listed gender, but no one will have more than one gender listed in the KB.
- Every account will be characterized with exactly one `account()` predicate and one `create()` predicate.

Show that your database works properly by formulating the following queries in Prolog (similar to what you did in the first assignment), and obtaining suitable answers:

(a) Is there an account in the Royal Bank of a man from Richmond Hill ?

(b) Is there a Canadian who has more than one account in CIBC?

(c) What are the banks in Toronto?

(d) What is a balance of an account in the Bank of Montreal of a person from Scarborough?

(e) What bank keeps accounts of at least two distinct local persons ? (Note: for purposes of this assignment, a person is *local* if it is a person who lives in Canada, and *foreign* otherwise.)

(f) What are the cities in the USA?

(These queries should *not* use English noun phrases!) You can use more queries, but at least all 6 queries mentioned above. It is up to you to formulate a range of queries that demonstrates that your program works properly. Keep your queries and answers computed by Prolog in the file `interaction.txt`

2 Helpers [6 marks]

In this part of this assignment, you should add to your KB *new rules* defining a few new concepts. When you will be developing your lexicon, you will need them to represent the meaning of the words that you will use in the lexicon. For example, add new rules defining predicates such as `city(X)`, `bank(B)`, `person(P)`, `man(M)`, `woman(W)`, `country(X)` in terms of the predicates given to you in Part 1. These predicates will be new concepts linked to English words described in the lexicon. The bodies of the rules can *use only the 5 predicates given above*. You should **not** add *new atomic facts with the new predicates*. Write these rules in Section `lexicon` of your file `bankAccount.pl`

3 Building a Lexicon [60 marks]

You will now build a lexicon, as we did in class, of articles, common nouns, proper nouns, adjectives, and prepositions to handle English noun phrases and the accounts they refer to. Each should be added to the lexicon section in `bankAccounts.pl` Here are some examples of the kinds of queries your system should be able to answer:

1. `what([a, city, in, canada], X).`
2. `what([the, canadian, man, with, a, large, account, in, a, local, bank], X).`
3. `what([any,foreign,male,person,with,a,small,account,in,a,canadian,bank], X).`
4. `what([a,foreign,male,person,from,losAngeles,with,a,small,account,in,rbc], X).`
   (note: for simplicity, we use “losAngeles” as if it is a single English word.)

5. `what([a, balance, of, a, large, account, in, a, local, bank], X).`

6. `what([any, local, bank, with, an, account, of, a, man, from, usa], X).`

7. `what([an, owner, from, canada, of, a, large, local, account], X).`

8. `what([a, woman, from, markham, with, a, medium, account], X).`

9. `what([a,bank,in,canada,with,a,small,account,of,a,foreign,person], X).`

10. `what([a, medium, account, in, a, canadian, bank, with, a, small, account, of, an, american], X).`

11. `what([the, balance, of, the, medium, account, in, metro_credit_union, of, a, woman, from, markham], X).`

12. `what([a,balance,of,an,account,of,an,american,with,a,small,account,in, a,local,bank,with,a,large,account],X).`

To cover the above examples your lexicon should include all the words mentioned above (in total, **25 words or more**, apart from articles). Notice that some words are ambiguous, and for this reason, you need several rules for them in your lexicon: one rule per possible meaning. (For comparison, consider the 4 rules for the preposition "with" discussed in class.) Remember that it is easy to defeat a language understanding program by using a word that it does not know about. Vocabulary is important in these systems. You will need at least the following words (but this list is not exhaustive). We will test all usages of the lexicon shown in the queries.

Article

Your lexicon should include the articles `a`, `an`, `the`, and `any`. For the purpose of this assignment, the meaning of these is all equivalent. For example, both [an, account, ...] and [a, account, ...] are correct.

Common Nouns

Your lexicon should include the common nouns `bank, city, country, man, woman, owner, person, account, balance`, and may be more. HINT: Once you have added some articles and common nouns, you can start testing your system with queries such as `what([a, bank], X)`, which should allow you to iterate over all the bank names by calling `More`. Note that `what([an, account], X)` should iterate over all account IDs mentioned in the KB.

Proper Nouns

Your lexicon should include all people names, bank names, IDs, country names and city names as proper nouns. We will be testing your system using our own knowledge base, so your rules should be able to handle having the knowledge base changed to other names. Any number (like month, year or amount) is also considered as proper nouns. You may use the library predicate `number(X)`, which is true if `X` is a number, for this purpose. Note that proper nouns are a referent to themselves and therefore, you can handle them with rules of the following form:

```
proper_noun(X) :- person(X). % X is the name of the person in this context
```

Adjectives

Your lexicon should define at least the following adjectives: `american, british, female, local, foreign, small, medium, large, old, recent, ...` Observe that some adjectives such as `canadian` are ambiguous, since they can be used to characterize a person, a bank, a city, an account, and so on. Note that while we may test your program with any countries, the only nationality adjectives you must handle are `canadian, american` and `british`. In addition, we clarify the following adjectives:

- `small` - a balance (or amount) is "small" if it is less than $1000.

- `large` - an amount is “large” if it is greater than $10000.
- `medium` - an amount is “medium” if it is gin between $1000 and $10000.
- `new` - an account is “new” (or “recent”) if it was opened in 2024. All other accounts are “old”.

Prepositions
Your lexicon should define the following prepositions: `of, from, in, with, ...`, where `of` means `ownedBy`. Observe that all these prepositions are ambiguous and your lexicon should have a separate rule per possible meaning of each preposition. You will lose marks if your lexicon will miss some of the intended meanings; however, expected meanings are shown in the examples above.

4 Testing the Lexicon [12 marks]

For this question, you will test your lexicon by trying queries using the `what` predicate and the simple noun phrase parser given in class. This parser has already been added to `bankAccounts.pl` file given to you.

Test the `what` predicate on a variety of noun phrases, like those above, showing that it is capable of identifying the entities being referred to by your noun phrases. It is up to you to choose noun phrases for testing, but you must convincingly demonstrate that your program works properly. Try at least **10 new** different noun phrases. Remember that testing your program is very important part of the software development cycle. You will lose marks, if you do not test your program as required.

Put your queries and a brief description (in plain English of what each is trying to test), in the file `testQueries.txt`. Do **not** include the 12 queries given to you in the previous section in this document. If you complete all or part of the next section, you can include your tests from that part in this file. Please note the following:

1. This part of your submission will not be evaluated on whether your system succeeds or not on your queries, but on the correctness of your queries. As such, we may test your knowledge base and queries on our own KB to evaluate if they are correct. However, when you design your KB in the first part of this assignment, try to include the facts that will help successfully retrieve the names referred by your queries.
2. As part of your interaction, you should test the output of your program when you ask for more solutions. It is fine, and generally expected, if you get duplicate answers. However, you should ensure that you only get correct answers. You will lose marks if your program produces any incorrect answers or misses any correct answers.

In addition, include the interaction you had with Prolog when using those 12 queries included in the previous section AND your new 10 queries in the file `interaction.txt`

5 Additional Features [12 marks]

Do this work only when the previous parts of your assignment are complete. The English noun phrases described above are quite limited. Generalize your program to handle some additional features of English:

- Handle the article “the”. The trick here is that a noun phrase like “the balance of the large account in cibc of a woman from markham” should succeed in naming a balance if there is a *unique* account of the appropriate kind. Otherwise, if there are several large CIBC accounts owned by women living in Markham, then the query should not retrieve a number. Another example: ”the balance of the largest account of robert in rbc” should succeed in naming the balance because even if robert has several accounts in rbc, only one of them has the largest balance. *Hint*: Update the parser section of `bankAccounts.pl` to add this functionality.
- Implement the adjectives `largest` and `oldest` which can be used to answer correctly queries like
`what[the, largest, account, of, a, woman, from, canada]`,
`what([the, oldest, account, in, metro_credit_union], A)`.
Hint: Update the parser section of `bankAccounts.pl` to add this functionality.

- Handle prepositional phrases of the form "between X and Y" applied to balances, e.g. `what([a, balance, between, 100, and, 25000], X)`. *Hint*: you can do this by modifying the parser, e.g., by adding there new rules that handle the words "between", "and". Also, you can use the built-in predicate `number(X)` that is true if X is a number.

6 Bonus work: LLMs for solving CSP problems [up to 10 points]

To make up for a grade on another assignment or test that was not what you had hoped for, or simply because you find this area of Artificial Intelligence interesting, you may choose to do extra work on this assignment. *Do **not** attempt any bonus work until the regular part of your assignment is complete.*

For this question, you will be testing the use of an LLM for generating Prolog code. To that end, take any one question from Assignment 3, and try to solve it using your LLM of choice.

Design a detailed prompt to an LLM. More specifically, start with an introduction saying that the LLM should imagine itself as an expert in Prolog coding and as an expert in solving Constraint Satisfaction Problems (CSPs). Include a small example of your choice of what is a CSP and what kind of Prolog program is expected as a solution to your example. Your example must be new and completely different from the questions included in Assignment 3. Request detailed comments from LLM and chain-of-thought reasoning that demonstrates how the LLM produced a program.

Write a report that is between a full page and two pages in length describing what version of LLM you used for testing, and the process you took. Use distinct fonts and different font colors for your writing and for output produced by an LLM. The LLM output should be clearly visible and distinct from your writing.

Your report should describe what the LLM got right, what it got wrong, and any adjustments you had to make to get it working/working better. You do not necessarily need to get the LLM to find a completely correct answer, as long as you make several attempts to improve its output and identify where it is wrong.

Additional notes:

- We will not accept any reports that are too long.
- While your report may contain some snippets of what the LLM produced, it cannot only contain LLM generated code. The main thing we are looking for is for you to document your experience, and analyze what went right/wrong.
- LLMs will often provide example output of running the program, but you should actually test the code yourself.
- You are not allowed to use the LLM to produce the analysis in the report itself. That must be your own work.

Submit your report in a PDF called `bonus.pdf` Remember you are only allowed to use an LLM for this question on the assignment, and only after you have initially completed all the other questions without help from an LLM. Doing otherwise will be pursued as "a breach of Policy 60: Academic Integrity." See the details in the CPS721 Course Management Form.