CPS721: Assignment 2

Due: October 8, 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 use ECLiPSE Prolog release 6 to mark the assignments. It is your responsibility to check that your code runs in ECLiPSE Prolog release 6. 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. You can test this by ssh-ing onto the department servers (*ie.* `moon`) and running ECLiPSE remotely from the command-line.

NEW FOR ASSIGNMENT 2: You are not allowed to use library predicates that we did not refer to in class, unless we specifically allow so below. For those we did cover in class, like `member` and `append`, you should re-implement yourself and use your own implementation. Make sure to use a slightly different name to avoid collision with the built-in predicate (ie. `myAppend` instead of `append`).

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, and are prohibited from using generative AI systems or online "help" websites for completing assignments. You may ONLY use a generative AI system for the bonus question (Q8), as described there. 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 `assignment2.zip`. It should contain 7 files:

`q1_list_equality.pdf`	`q2_equal_entries.pl`	`q3_alternate.pl`
`q4_nested_find.pl`	`q5_robocup.pl`	`q6_list_shift.pl`
`q7_preorder.pl`	`(optional) q8_report.pdf`	

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. Do not submit `.rar`, `.tar`, `.7zip`, other compression format aside from `.zip`, or submit multiple `.zip` files. If you do so, you will lose marks. All submissions should be made on D2L. Submissions by email will not be accepted.

As long as you submit the file with name `assignment2.zip`, you may 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.

1 List Equality [20 marks]

For each of the following pairs of lists, state which can be made identical and which cannot. You must also provide a short proof as to why. This means you should convert the lists to the same style (*ie.* the '`|`' based representation or the standard ',' based representation), and use that to explain how they can be made identical, element by element, or not. For example, if given the pairs `[X, Y]` and `[a | [b]]`, you could say that these match with `X = a` and `Y = b` since the second list can be written as `[a, b]` (or equivalently the first list can be written as `[X | [Y]]`). Any answers that do not contain detailed explanations for why the lists do or do not match will lose marks.

You should submit your answers in a pdf file called `q1_list_equality.pdf`. The names, emails, and student IDs of your group members should appear at the top of this PDF file.

a. (2 marks) `[X, Y | Z]` **and** `[a, b, c | [d, e, Y] ]`

b. (2 marks) `[q, [A | [r, s] ], t]` **and** `[q, [r, [r, s] ] | B]`

c. (2 marks) `[ [Cow | [cat, dog] ], bird, bug, chicken ]` **and**
`[ [ant, [cat, dog] ] | Horse]`

d. (2 marks) `[1, A, 2 | [A, 3, 4] ]` **and** `[B | [2, C | [D | E ] ] ]`

e. (3 marks) `[A | [ A | [ [ A | [ [A] ] ] ] ] ]` **and** `[b | C]`

f. (3 marks) `[X | [Y | [ Z | [X] ] ] ]` **and** `[all, around, the, world, Y]`

g. (3 marks) `[1, 2 | [ X | [ Y, Z | X] ] ]` **and** `[Q | [R, S, [], [ [Y] ] ] ]`

h. (3 marks) `[Lions, [[and], tigers], [and], bears, oh | [[my]] ]` **and**
`[[I, have], [[A], Bad], Feeling | [About | This] ]`

2 Finding Equal Entries [5 marks]

Write a program called `equalEntries(List1, List2, EqualItems)`, which takes in two lists as given input and returns a boolean list which indicates which of the indices the two lists are equal on. That is, if the lists agree on their third elements, then the third element of `EqualItems` will be `true`. Otherwise, it will be `false`. Below you can find example queries when using this predicate:

```
?- equalEntries([a, b, c, d, e], [1, 2, c, 4, 5], [false, false, true, false, false]).
    Yes

?- equalEntries([a, b, [c], d, [e, f, g]], [1, b, c, d, [e, f, g]],
                  [false, true, false, true, true]).
    Yes

?- equalEntries([a, b, [c], d, [e, f, g]], [1, b, c, d, [e, f, g]], X).
    Yes with X = [false, true, false, true, true]

?- equalEntries([a, b, [c], d, [e, f, g]], [1, b, c, d], X).
    No
```

The last one fails since the input lists have a different size. Note that only `EqualItems` may be set as a variable when calling this program. The input lists will also not contain any variables.

Write your program in the file called `q2_equal_entries.pl` in the `equalEntries` section. You are allowed to include any helper predicates you see fit.

3 Alternating Adding and Subtracting [10 marks]

Write a program called `alternatePlusMinus(List, Sum)` which takes in a list of integer numbers and alternates adding and substracting the values. Below you can find example queries when using this predicate:

```
?- alternatePlusMinus([1, 2, 3], 0).
    Yes
```

This holds because $1 + 2 - 3 = 0$.

```
?- alternatePlusMinus([5, 6, -1, -2, 7, 8, 9], Z).
    Yes with Z = 2
```

This holds because `5 + 6 - (-1) + (-2) - 7 + 8 - 9 = 2`.

Notice that in both cases, the first operation is a “+”. In addition, the result of calling your program on an empty list should be 0.

Write your program in the file `q3_alternate.pl` in the `alternatePlusMinus` section.

HINT: Create a helper predicate which keeps track of whether the next step is a “plus” step or a “minus” step.

4 Find in Nested List [15 marks]

In this question, you will be creating different programs for considering the contents of nested lists.

a. [**10 marks**] Write a program called `nestedFindDepth(List, Item, Depth)`, which takes in a `List` of nested lists and an `Item`, and returns the depth that `Item` appears in the `List`.

Below you can find example queries when using this predicate:

```
?- nestedFindDepth([[a, b], [[c]], d], c, 2).
    Yes

?- nestedFindDepth([[a, b], [[c]], d], b, D).
    Yes with X = 1

?- nestedFindDepth([[a, b], [[c]], d], [a, b], D).
    Yes with D = 0
```

Note that in all tests, the input list will not contain any variables. However, either or both of `Item` or `Depth` may be a variable. When doing so, your program should be able to return all correct answers (and no incorrect answers) when more solutions are requested. For example, if the depth is set to 2 and `Item` is set as a variable, then your program should return all elements at depth 2 when multiple solutions are requested.

Write your program in the `nestedLists` section of `q4_nested_find.pl`. You may use the built-in predicate `is_list/1`, which holds if the given argument is a list. You may also define your own helper predicates. They should be included in the same section.

b. [**5 marks**] Write a program called `nestedFindIndex(List, Item, Depth, Index)`, which takes in a `List` of nested lists and an `Item`, and returns the depth that `Item` appears in `List` and the `Index` in the `List` that it appears in the top-level list. Below you can find example queries when using this predicate:

```
?- nestedFindIndex([[a, b], [[c]], d], b, 1, 0).
    Yes

?- nestedFindIndex([[a, b], [[c]], d], c, D, Y).
    Yes with D = 2 and Y = 1

?- nestedFindIndex([[a, b], [[c]], d], E, D, 2).
    Yes with E = d and D = 0
```

The first query holds because `b` appears in the 0th element, namely `[a, b]`, in the given input list. The other queries hold similarly.

The input list will always be a list that does not contain variables. As such, `nestedFindIndex(a, E, X, Y)` should fail, since `a` is not a list and therefore does not have any elements at different indices. Moreover, any combination of the arguments other than the list may be set as a variable. When doing so, your program should be able to return all correct answers when more answers are requested.

Write your program in the `nestedLists` section of `q4_nested_find.pl`. You may use the built-in predicate `is_list/1`, which holds if the given argument is a list. You also define your own helper predicates and re-use your answer from part **(a)** if you choose.

5 Robocup with Lists [20 marks]

Recall the Robocup question from Assignment 1.[1] There, you were asked to find if it was possible to score in at most a given number of passes/shots. However, your predicates in Assignment 1 only indicated whether it was possible, and they did not provide information about what path of passes and shots were necessary. In this question, you will use lists to compute such information, as well as to ensure that there are no unnecessary passes (ie. the ball never reaches the same robot more than once). That is, you will create two new versions of the `canPass` and `canScore`:

- `canPass(R1, R2, M, Path)` - this predicate is true if robot `R1` has the ball, they can get the ball to `R2` in *at most* `M` passes, and `Path` is the list of robots along the way from `R` to `R2`.
- `canScore(R, M, Path)` - this predicate is true if `R` can score using *at most* `M` kicks, where a kick is either a pass or a shot at the net. **This means that `R` has a clear path to kick directly at the net, and the robot with the ball can pass it to `R` in at most `M - 1` passes.** `Path` is the list of robots along the way from the robot that has the ball to `net`.

To better understand what is required, recall the situation given in assignment 1:

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

Here, `canScore(r3, 1, [r3, net])` holds since `r3` has the ball, and the path is clear between `r3` and the `net`. It also holds that `canPass(r3, r1, 1, [r3, r1])`, since the path since the path is clear between `r3` and `r1`. Since the path is clear from `r1` to the `net`, this also means that `canScore(r1, 2, [r3, r1, net])` holds, as does `canScore(r1, 3, [r3, r1, net])`. ~~In addition, since r3 can get the ball r2 with one pass, who can then pass it to r1, it is also true that canScore(r2, 3, [r3, r2, r1, net])~~.

Additional notes:

- For `canPass`, the first element of `Path` should be `R1` and the last should be `R2`. For `canScore`, the first element of `Path` should be the robot with the ball, and the last one should be `net`.
- For both `canPass` and `canScore`, there should not be any robots that appear more than once along `Path`. For example, `Path` cannot equal something like `[r3, r1, r3]` for `canPass` or `[r3, r1, r3, net]` for `canScore`.
- Recall that `pathClear` is a symmetric relation, a robot cannot pass to the `net` (or vice versa), and a robot cannot pass to itself.
- You do **not** need to submit the knowledge base you use for testing. Space has been made for it in the required file, but we will not be marking it. That space is simply there to make it easier for you to test your program. It is recommended that you test with other KBs to ensure your program's correctness.
- Your program should be able to handle queries like `canScore(R, 3, Path)` which can be read as find me a robot who can score in at most `3` kicks. By asking for "More", you should be able to get all robots who can score in at most 3 kicks. Your program should similarly be able to handle queries like `canPass(R1, R2, 3, Path)` which can be used to find all pairs of robots that can pass between each other in at most 3 passes. However, the maximum value `M` will not be set to a variable in all test queries.

Write your program in the `robocup` section of the file `q5_robocup.pl`. You may also define and use any helper predicates that you wish, but they must be in this section of the file.

[1]This question was updated on September 29 due to an important clarification. See the **bold** and ~~strikethrough~~ text.

6 List Shifting [15 marks]

For this question, you will use the term `next(Head, Tail)` to represent a Prolog list `[Head | Tail]`. Here, we will use `nil` to represent the empty list `[]`. For example,
`next(7, next(1, next(5, next(0, next(9, nil)))))` represents the list `[7,1,5,0,9]`.

You should now write a program that takes a list written as a term, and shifts a given number of elements from the beginning of the list to the end of the list. For example, if you are asked to shift the above example by 2, the first two elements (namely 7 and 1) should be moved to the end of the list and the result would be

```
next(5, next(0, next(9, next(7, next(1, nil)))))
```

The name of the predicate should be `listShift(List, V, Result)` where `List` is the input list, `V` is the number of items to shift by, and `Result` is the result after the shift. You can assume that `V` is a non-negative integer. The following shows several examples

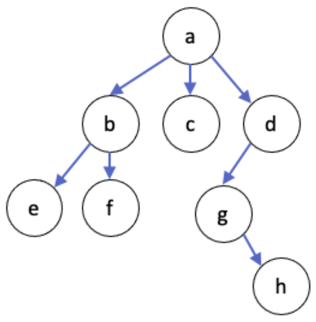
```
?- listShift(next(7, next(1, next(5, next(0, next(9, nil))))), 2, X).
       Yes with X = next(5, next(0, next(9, next(7, next(1, nil)))))

?- listShift(next(7, next(1, next(5, next(0, next(9, nil))))), 3, X)
       Yes with X = next(0, next(9, next(7, next(1, next(5, nil)))))

?- listShift(next(7, next(1, next(5, next(0, next(9, nil))))), 20, X)
       Yes with X = next(7, next(1, next(5, next(0, next(9, nil)))))
```

Notice in the last example that if `V` is larger than the length of the list, the whole list is shifted to the end, and so nothing changes.

Write your program in the `listShift` section of `q6_list_shift.pl` file. You may introduce any helper predicates that you choose. We will not test your program with `List` or `V` being set as a variable. Only `X` may be set as a variable. You are NOT allowed to use any standard Prolog list notation as part of your solution.

7 Preorder Tree Traversal [15 marks]

Recall that a preorder traversal of a tree means that a node is visited prior to its children. For example, consider the following ternary tree, which is a tree where each node has at most 3 children (much like how a binary tree has at most two children at every node):



A preorder traversal would mean that the nodes are visited in the following order: a, b, e, f, c, d, g, h. In this question, you will write a program that takes in a ternary tree and generates the list which corresponds to the preorder traversal of the tree. To represent a ternary tree, we will use the following term `tree3(Name, Left, Middle, Right)`. Here, `Name` is the name of the node, and `Left`, `Middle`, and `Right` represent the three branches of the tree. The values of each these will be terms: either a `tree3` term or `none` indicating there is no branch below.

Lists will be represented using the term `next(Head, Tail)` as in the previous question. Your program for listing the tree elements as they would be in a preorder traversal should then be defined as follows:

`preorder(Tree, List)` - where `Tree` is the given tree and `List` is the list of elements as they appear during the preorder traversal.

For example, if your program is called as `preorder(Tree, X)` given the tree above, then your program should succeed with

```
X = next(a, next(b, next(e, next(f, next(c, next(d, next(g, next(h, nil))))))))
```

In addition, note the following:

- We will not test your program with `Tree` as a variable.
- If `Tree = none`, then your program should succeed with the list `nil`.

You should write your program in `preorder` section of the file `q7_preorder.pl`. You may add helper predicates as you see fit. The tree above, and a few other trees, have already been included in that file for testing purposes. You may want to test on additional trees to ensure your program works in all cases. Note that we have used an extra predicate in the file to make testing easier. See the file for more information. Make sure you do not put any such trees in the `preorder` section of the given file. You are NOT allowed to use standard Prolog list notation in your solution program.

8 BONUS: LLMs for Prolog Coding [10 marks]

ONLY COMPLETE THIS QUESTION IF YOU HAVE COMPLETED THE REST OF THE ASSIGNMENT. YOU ARE ONLY ELIGIBLE FOR BONUS MARKS IF GET AT LEAST 75% ON EACH OF THE QUESTIONS 1 THROUGH 7. IF YOU DO NOT DO THIS QUESTION, DO NOT SUBMIT ANY PDF FOR THIS QUESTION.

For this question, you will be testing the use of an LLM for generating Prolog code. To that end, take either Q2 or Q3 from **Assignment 1** (but not both), and try to solve it using your LLM of choice. Write a report that is between half a page and a full page in length describing what LLM you used for testing, and the process you took. 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. This may include asking for additional queries or trying additional examples.
- Recall that we had additional requirements on what operators you could or could not use. You should try to ensure the LLM follows those restrictions as well.
- 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 `q8_report.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."