CPS721: Assignment 2

Due: October 10, 2023, 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. By submitting this assignment, you acknowledge that you have read and understood the course policy on collaboration as 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 `assignment2.zip` containing 6 files:

```
q1_list_equality.pdf
q2_increasing_power_sum.pl   q3_zipper.pl            q4_deepest_nesting.pl
q5_partition_months.pl       q6_collab_distance.pl
```

The latter 5 files have been given to you and you should fill them out using the format described. 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 `assignment2.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 on 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.

LIST PROGRAM QUESTION POLICY: For questions 2 through 9, you are supposed to write programs to solve a variety of problems. Please submit these by modifying the provided files and submitting them. Ensure you follow the required format, where code is to be put in the correct section. Failing to do so will lose you marks.

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 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) `[W, X, Y | Z]` **and** `[1, 2, 3, 4, 5 | [6, 7, X] ]`

b. (2 marks) `[p | [q | [ r | [s | [ t | [V] ] ] ] ] ]` **and** `[X, Y| Z]`.

c. (2 marks) `[ [Z | [x, y] ], e, f, g ]` **and** `[ [a, [x, y] ] | V]`

d. (2 marks) `[ [a], B, C | D]` and `[ [a | [B] ] | [C | D] ]`

e. (3 marks) `[minus | [Y, X | [ minus, Y | [X] ] ] ]` and
`[X, plus, minus | [X, Y, minus] ]`

f. (3 marks) `[bike | A]` and `[C | [C | [C | [C | [C] ] ] ] ]`

g. (3 marks) `[a, b | [ C | [ D, E | C] ] ]` and `[F | [G, H, [], [ [D] ] ] ]`

h. (3 marks) `[Fox, [[in], socks], [on], box, on | [[knox]] ]` and
`[[The, cat], [[in], The], Hat | [Comes | Back] ]`

2 Increasing Power Sum [10 marks]

Write a program called `increasingPowerSum(List, Power, PowerInc, Sum)` where `List` is a list of integers, and `Power` and `PowerInc` are both non-negative integers. The value of `Sum` should be the sum of the integers in the list, each taken by a different power starting with `StartingPower` and incremented by `PowerInc`. This means that `Sum` will be given by the sum of the first element of the list to the exponent of `Power`, plus the second element of the list to the exponent of `Power + PowerInc`, plus the third element of the list to the power of `Power + 2 * PowerInc`, etc.

Below you can find examples of queries using this predicate.

```
?- increasingPowerSum([1, 2, 3, 4], 1, 2, 16636).
    Yes
```

This holds because $1^1 + 2^3 + 3^5 + 4^7 = 16636$

```
?- increasingPowerSum([5, 4, 3], 1, 0, X).
    Yes with X = 12
```

This holds because $5^1 + 4^1 + 3^1 = 12$

Note that `X^Y` computes `X` to the exponent `Y`. Your program should return 0 if it is given an empty list. We will not test with enormous numbers or negative powers to avoid overflow or floating point issues. You can assume that whenever your program is called, `List`, `Power`, and `PowerInc` will not be variables.

Submit your program in the file `q2_increasing_power_sum.pl`.

3 Zipper Lists [15 marks]

Write a program, `zipper(List1, List2, Zipper)` which takes in two lists, `List1` and `List2`, and creates a new list called `Zipper` given by alternating the elements in the first two lists. The alternation should start with an element from `List1` (assuming one exists). If one list runs out of elements before the other, then the remainder of the list with elements should be added to the end of `Zipper`.

The following are examples of expected behaviour:

```
?- zipper([1, 2, a, b], [f, g, c, d], [1, f, 2, g, a, c, b, d]).
Yes

?- zipper([a, b, c, d], [1, 2], Zipper).
Yes with Zipper =  [a, 1, b, 2, c, d]
```

Hint: use a helper predicate which keeps track of which list to take from next. We may test your program with any of the arguments set as a variable, but only a single argument will be set as a variable at any time.

Submit your program in the file `q3_zipper.pl`.

4 Deepest Nested List [15 marks]

Write a program `deepestNesting(List, Depth)`, which takes in a list of nested lists, and returns the depth of the deepest element in the list.

The following are examples of the expected behaviour:

```
?- deepestNesting([ [], abc, 1, [6, 7, [8] ] ], 3).
Yes
```

This holds because `8` is inside list `[8]`, which is inside list `[6, 7, [8] ]`, which is inside the original list.

```
?- deepestNesting([ [a, [ [ [] ] ] ], [j], b], Depth).
Yes with Depth = 4
```

This holds because of the inner most empty list `[]`. Since it does not contain an element, the empty list itself is the deepest element. This means that `deepestNesting([], 0)` and `deepestNesting(a, 0)` both hold.

You may use the built-in predicate `is_list/1`, which holds if the given argument is a list. You may also find the built-in predicate `max/3` useful for this question. Finally, note that only the `Depth` argument may be set as a variable in our tests.

Submit your program in the file `q4_deepest_nesting.pl`.

5 Stock Price Tracker [15 marks]

For this question, you will create a program that takes in a list of months and the price of some stock at the end of each of those months, and your task is to partition the months by whether or not the stock increased or decreased over the previous month. Specifically, you should write the program `partitionMonths(Months, Prices, Increase, Decrease)` where `Months` is a list of months, `Prices` is the stock price at the end of each of those months, `Increase` is the list of months in which the stock price increased or stayed the same compared to the previous month, and `Decrease` is the list of months in which the stock price decreased when compared to the previous month.

The following are examples of the expected behaviour:

```
?- partitionMonths([jan, feb, mar, apr, may, jun, jul],
          [7.90, 8.00, 8.15, 8.05, 8.04, 8.05, 8.5], [feb, mar, jun, jul], [apr, may]).
Yes

?- partitionMonths([january, february, march, april],
          [56.20, 56.25, 56.25, 56.50], Increase, Decrease).
Yes with Increase = [february, march, april] and Decrease = []
```

Notice that you should not make any assumptions on the names used for months. Your program should fail if the `Month` and `Prices` list do not have the same length, or if they contain less than two items each (since we need two months to say if a price has increased or decreased). Only the `Increase` and `Decrease` arguments may be set as a variable in our tests.

Submit your program in the file `q5_partition_months.pl`.

6 Collaboration Distance with Lists [25 marks]

Recall the collaboration distance question from Assignment 1. In that version of the problem, you determined if there was a collaboration path no longer than a given value between two authors. However, the path could repeat authors (or articles) and the result of a query didn't clearly show any information about what authors (or articles) were along the collaboration path. In this question, you are going to address these limitations using lists.

Using the `articleAuthor` predicate from Assignment 1, create a program `collabDist(Author1, Author2, MaxDist, Authors, Articles)` that determines if there is a collaboration path between `Author1` and `Author2`, and returns the authors and articles along that path as lists called `Authors` and `Articles`, respectively. Moreover, both `Authors` and `Articles` should **not** contain an element more than once.

For example, suppose the knowledge base has 3 authors (`tom`, `jennifer`, and `tina`) and two articles (`a1` authored by `tom` and `jennifer`, and `a2` authored by `jennifer` and `tina`). Then `collabDist(tom, tina, 2, Authors, Articles)` should succeed with `Authors = [tom, jennifer, tina]` and `Articles = [a1, a2]`.

Additional notes:

- Notice that the first element of `Authors` should be `Author1` and the last should be `Author2`.
- If `Author1 = Author2`, then the query should succeed for any `MaxDist` value that is non-negative provided the author has written at least one article. In that case, the author list should consist solely of one author (and have a length of 1) and the article list should contain exactly one article.
- You do **not** need to worry about the topic of the paper. You only need to complete `collabDist`, **not** `collabDistWithAI`. Your test knowledge bases thus also do not need to use `articleTopic`.
- 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.

Your program should also handle queries like `collabDist(tom, G, 2, Authors, Articles)`, which can be read as find me an author, `G`, who is no more than a collaboration distance 2 away from `tom`. If called with `;` or `More`, this should return all authors that are a collaboration distance of no more than 2. To do so, be careful about the interplay between `not` and `=`. You will still get part marks if your program works when given two authors, even if it can not handle when one of the authors is a variable.

Submit your program in the file `q6_collab_distance.pl`.