cps721: Assignment 3 (100 points).
Due date: Electronic file - Thursday, October 24, 2019, 21:00 (sharp).
YOU SHOULD NOT USE ";" , "!" AND "–>" IN YOUR PROLOG RULES.

> You MUST work in groups of TWO, or THREE, you cannot work alone. You can discuss this assignment only with your CPS721 group partners or with the CPS721 instructor. 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.

This assignment will exercise what you have learned about constraint satisfaction problems (CSPs). In the following questions, you will be using Prolog to solve such problems, as we did in class. For each of the questions, you should create a separate file containing rules for the `solve(List)` predicate and any other predicates you might need. Note that it is your programs that have to solve each of the problems in this assignment, not you! You lose marks, if you attempt to solve a problem (or any small part of the problem) yourself, and then hack a program that prints a solution. All work related to solving a problem should be done by **your Prolog program**, not by you.

1 (40 points). Use Prolog to solve the following crypt-arithmetic puzzle involving multiplication and addition:

```
          G E T
  *         B Y
      ---------
       B A B E
  +    G E T
     -----------
    B E A R E
```

In other words, $Y * GET = BABE$, $B * GET = GET$, $BABE + GET * 10 = BEARE$. Assume that each letter stands for a distinct digit and that leading digits are not zeroes (do not try to guess them yourself!).

First, you can try to solve this problem using pure generate and test technique, without any interleaving, and notice how much time it takes. (Warning: it can take several minutes depending on your computer.) Determine how much computer time your computation takes using the following query:

```
?- X is cputime, <your query>,  Y is cputime, Z is Y - X.
```

The value of `Z` will be the time taken. Keep this "pure generate and test" version of your program in the file `puzzle1.pl` You lose marks if you do not provide a working version of this program.

Next, solve this problem using smart *interleaving of generate and test* approach, as discussed in class. Keep this program in the file `puzzle2.pl` and make sure that the TA can compile this file, and run this program using either the main predicate **solve(L)** or **print_solution(List)**. Write comments in your program file: explain briefly the order of constraints you have chosen and why this has an effect on computation time. You can draw a dependency graph by hand (upload a PDF file with an image) or using ASCII pseudo-graphics (in your file), if you decide to use one. Find also how much time your program takes to compute an answer, but do not include printing time.

For both programs, write your session with Prolog to the file **puzzle.txt**, showing the queries you submitted and the answers returned (including computation time). Make sure that your output is easy to read. Print your solution using the predicate `write(X)` and the constant `nl`, similar to the `print_solution` predicate that we considered in class, but make sure your rule for the predicate **print_solution(List)** takes a list of variables as an argument.

Handing in solutions: (a) An electronic copy of your files **puzzle1.pl** **puzzle2.pl** and **puzzle.txt** must be included in your **zip** archive.

Part 2 (20 points). The members of a sport club were electing new officers to the Executive Council. The positions of president, vice-president, treasurer and secretary were open for election. There were six candidates for these four positions: Arthur, Bart, Colleen, Donna, Eva, and Frank. Election turned out to be difficult because candidates had lots of complicated preferences.

1. Arthur did not want to work without Bart, but in any case he did not want to run for the vice-president position.
2. Bart agrees to serve as one of the officers but neither as the vice-president nor as the secretary.
3. Colleen did not like working with Bart, unless Frank would be also elected.
4. Donna was strictly against working either with Eva, or with Frank.
5. Eva said she is not going to serve if both Arthur and Bart would be elected.
6. Frank can agree to serve only as the president, and under an additional condition that Colleen is not elected as the vice-president.

Despite these complications, an Executive Council can be successfully elected: there exists only one solution to this set of constraints.

Your task is to write a PROLOG program using the smart interleaving of generate-and-test technique explained in class: your program has to compute who was elected and to what position. Do not attempt to solve any part of this puzzle yourself, i.e., do not make any conclusions from the statements given to you. To get full marks, you have to follow a design technique from class. However, you may introduce your own helping predicates to formulate constraints correctly. For example, you might wish to introduce the new helping predicate *incompatible(X,Y,Z)* to say that candidates X, Y, Z cannot be elected altogether. It is up to you how many helping predicates you need, but make sure you implement them correctly. Also, explain briefly what they mean and why do you need them. You cannot use any of the library predicates, unless they were introduced in class. Notice there are some hidden constraints that are not stated explicitly. Nevertheless, they must be formulated and included in the program to find a correct solution satisfying all explicit and implicit constraints. Be careful in your program regarding the order of constraints. Explain briefly the ordering you have chosen (write comments in your program which constraint you implement where). You lose marks if you do not explain. Make sure you print clearly the solutions: you lose marks if output is difficult to read. *Hint: as a finite domain for your variables you can choose values 0 (not elected), 1 (president), 2 (vice-president), 3 (treasurer) and 4 (secretary).*

Handing in solutions: (a) An electronic copy of your file **sportClub.pl** must be included in your **zip** archive; (b) your session with Prolog, showing the query `solve(List)` and the query `print_solution(List)` that you submitted and the answers returned (both the query and the answers must be in comments inside the file **sportClub.pl**); You should also include computation time. Make sure that your answers are easy to read.

Part 3 (40 points).

In a tournament, five teams played five rounds of a game. In each round, one of the teams did not participate, while the other four teams played two matches once. In each match, two teams play against each other. In total, the tournament had 10 matches. Points for each match are awarded as follows: a team is awarded 2 points for a win, 0 points for a loss, and 1 point for a draw. The teams are from Oakville, Pickering, Richmond Hill, Scarborough, and Toronto (downtown).

Your task is to write a Prolog program that finds points awarded per match to each team and the total number of points per team. You have to write a Prolog program that implements precisely not only the given constraints, but also constraints that remain implicit. Never try to guess part of solution by yourself: all reasoning should be done by your program. You have to demonstrate whether you learned well a program design technique. Your implementation

should follow exactly one of the design patterns explained in class. You lose marks, if the TA will see that you embed your own reasoning into your program. Points per match must satisfy all of the following constraints.

1. Pickering lost to Scarborough in the first round, but won over Oakville in the second round.
2. Toronto did not play in the third round; they had one win and one loss in the previous two rounds.
3. Oakville did not participate in the fourth round, but they already won twice in the preceding three matches.
4. All matches in the fourth and in the fifth round finished with a draw.
5. Before the fourth round, Richmond Hill won only once and lost once.
6. None of the matches in the first, in the second and in the third rounds finished with a draw.

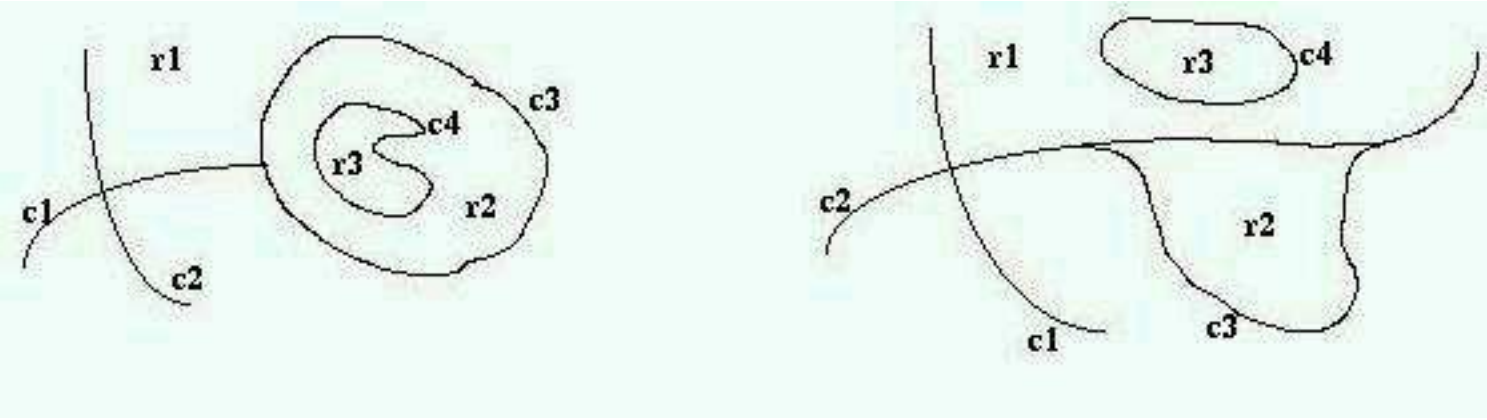
You will need to be careful regarding the order of constraints. Explain briefly the ordering you have chosen (write comments in you program). Also, provide comments where in your program which constraint is implemented. Make sure you print clearly the solutions: you lose marks if output is difficult to read. Arrange your output similar to the predicate *print_solution(List)* discussed in class. The TA should be able to call both your predicate *solve(List)* and your predicate *print_solution(List)* to see the solution (there is only one correct solution).

Hint: you might wish to consider the 2-argument predicate *fourExactly(X,List)* as a helping predicate and use it in your program. This predicate is true, if *X* occurs exactly four times in the given *List*. This predicate can be useful when you implement the 4th constraints: the list of points awarded to the teams has exactly 4 occurrences of "1". You might wish to introduce additional helping predicates to implement constraints. In any case, you have to implement all your helping predicates yourself. You cannot use any of the library predicates, unless they were discussed in class. In your programs, the variables should range over the following finite domain of integers: -1 (a team did not participate in a round), 0 (loss), 1 (draw), 2 (win).

Handing in solutions. (a) An electronic copy of your program (**tournament.pl**) with all defined predicates (you must also provide brief comments); (b) your session with Prolog (including time taken), showing your query and the clearly printed answers (keep your session inside comments in the same file **tournament.pl**) .

4 Bonus work (40 points). To make up for a grade on another assignment that was not what you had hoped for, or simply because you find this area of AI 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. Bonus work is individual.*

This question refers to an instance of the problem of *scene interpretation*, a problem in computer vision. Specifically, we would like to write a Prolog program that will help us interpret sketch maps of the type depicted below.



In both maps, there are two types of entities: (a) *chains*, corresponding the (curved) line segments in the map; and

(b) *regions*, corresponding to the 2D regions that are bounded by these chains. These two maps have four chains and three regions (region r_1 is outermost region encompassing the entire scene). These chains and regions stand in various relations to one another:

- two chains can *cross* one another (c_1 and c_2 on both maps)
- one chain can *join* another by forming a "tee". On the left map, c_1 joins c_3, but not vice versa; on the right map c_2 joins c_3 and c_3 joins c_2.
- a chain can be a *loop* (on the left map, both c_3 and c_4 are loops).
- a chain can be *beside* a region (on both maps c_1 is beside r_1, while on the left map c_3 is beside r_1 and r_2).
- a region can be the (immediate) *interior* of a loop (on the left map, r_2 is interior of c_3 and r_3 is interior of c_4).
- a region can lie in the (immediate) *exterior* of a loop (on the left map, r_1 is exterior of c_3, r_2 is exterior of c_4).

Our ultimate job is to come up with a reasonable interpretation of maps by *labeling* each chain and region. Specifically, each chain is either a *road*, a *river*, or a *shore*; and each region is either *water* or *land*. A labeling assigns to each chain and region *one* such label. Not all labellings are valid. Here are some constraints that we use to specify valid labellings:

1. No river can cross another river; a chain cannot cross a shore.
2. If a river joins another "chain", that chain must be another river or a shore. A road cannot join a river.
3. A river cannot be a loop.
4. A shore must be a loop (so our maps must be large enough to include entire lakes).
5. A river or a road can only be beside land, not water.
6. If a chain is a shore, then either its interior region is water and its exterior is land, or its exterior region is water and its interior is land.

You are to implement a small Prolog program that determines a valid labeling. You can use Prolog code in the file **part4.pl** that is provided together with this assignment. Input to the program is small knowledge base (KB) that provides the data specifying the scene. An example (in the file **left.pl**) for the left map is:

```
crosslist([ [c1,c2], [c2,c1] ]).
joinlist([ [c1,c3] ]).
looplist([ [c3,c3], [c4,c4] ]).
besidelist([ [c1,r1], [c2,r1], [c3,r1], [c3,r2], [c4,r2], [c4,r3] ]).
insidelist([ [c3,r2], [c4,r3] ]).
outsidelist([ [c3,r1], [c4,r2] ]).
```

The predicate `crosslist` is true of the list of all pairs of chains that cross one another, where each pair is itself represented as a list. The predicate `joinlist` is true of the list of all pairs of chains such that the first chain in the pair joins the second. The predicate `looplist` describes the list of chains that are loops (note that this list also contains pairs of chains, but elements in every pair are identical). The predicate `besidelist` describes chain-regions pairs such that the chain is beside the region. The predicate `insidelist` specifies all chain-region pairs where the region is the interior of the chain (which must be a loop), and `outsidelist` does the same for exteriors. In addition, there are two predicates that describe domains: `chain(C,LX)` is true if a chain C is labeled with label LX, and `region(R,LY)` is true if a region R is labeled with label LY.

To solve this problem, the program chooses labels for all scene elements, converts lists given in the KB to the corresponding lists of labels and then tests whether labels are valid with respect to all constraints. For example, to test whether labels for elements in the `crosslist` are valid, the program calls the predicate `crossListValid(Labels).` To give a sense of the structure of constraints, the program includes a partial implementation of this predicate.

Your task is to complete the rules that define predicates:

```
crossListValid(CrossListOfLabels)
besideListValid(BesideListOfLabels)
joinListValid(JoinListOfLabels)
loopListValid([ChainLab1,ChainLab2,ChainLab3,ChainLab4], LoopListOfLabels)
listInsideOutsideValid(InLabels,OutLabels)
```

Implement these and all remaining predicates in Prolog, and test them on two scenes provided in the files **left.pl** and **right.pl** (keep these two KBs in separate files and load only one of the two files at a time when you do testing). Your program will be tested on another scene (not given to you). If predicates in your program are not consecutive, read the handout regarding what compiler directive you should include.

Handing in solutions. An electronic copy of: (a) your working program (**vision.pl**) with all defined predicates (you also must provide brief comments in the program); (b) your session with Prolog, showing the queries you submitted and the answers returned (the name of the file must be **vision.txt**). Request all correct interpretations for both given maps (using ";"). Explain briefly why you are getting different interpretations and if they are intuitively valid. *If you work in a group, write clearly the name of the person who submits bonus work. Otherwise, the marks will be equally divided between all students in your group.*

How to submit this assignment. Read regularly *Frequently Answered Questions* and replies to them that are linked from the Assignments page at

`http://www.scs.ryerson.ca/~mes/courses/cps721/assignments.html`

If you write your code on a Windows machine, make sure you save your files as plain text that one can easily read on Linux machines. Before you submit your Prolog code electronically make sure that your files do not contain any extra binary symbols: it should be possible to load either `puzzle2.pl` or `sportClub.pl` or `tournament.pl` into a recent release 6 of ECLiPSe Prolog, compile your program and ask testing queries. TA will mark your assignment using ECLiPSe Prolog. If you run any other version of Prolog on your home computer, it is your responsibility to make sure that your program will run on ECLiPSe Prolog (release 6 or any more recent release), as required. For example, you can run a command-line version of *eclipse* on moon remotely from your home computer to test your program (read handout about running *ECLiPSe Prolog*). To submit files electronically do the following. First, create a **zip** archive:

```
zip yourLoginName.zip puzzle1.pl puzzle2.pl puzzle.txt sportClub.pl tournament.pl
```

where `yourLoginName` is the Ryerson login name of the person who submits this assignment from a group. Remember to mention at the beginning of each file *student, section numbers* and *names* of all people who participated in discussions (see the course management form). You may be penalized for not doing so. Second, upload your file

yourLoginName.zip

(make sure it includes **all** files) to D2L into "Assignment 3" folder.

Improperly submitted assignments will **not** be marked. In particular, you are **not** allowed to submit your assignment by email to a TA or to the instructor.

Revisions: If you would like to submit a revised copy of your assignment, then run simply the submit command again. (The same person must run the submit command.) A new copy of your assignment will override the old copy. You can submit new versions as many times as you like and you do not need to inform anyone about this. Don't ask your team members to submit your assignment, because TA will be confused which version to mark: only one person from a group should submit different revisions of the assignment. The time stamp of the last file you submit will determine whether you have submitted your assignment on time.