cps721: Assignment 3 (100 points).
Due date: Electronic file - Monday, October 24, 2022, 21:00 (sharp).

You have to work in groups of TWO, or THREE. You should not work alone.

YOU SHOULD NOT USE ";" , "!" AND "–>" IN YOUR PROLOG RULES.
ALSO, YOU CANNOT USE SYSTEM PREDICATES OR LIBRARIES NOT MENTIONED IN CLASS.

> You can discuss this assignment only with your CPS721 group partners or with the CPS721 instructors. 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. By submitting this assignment you acknowledge that you read and understood the course *Policy on Collaboration* in homework assignments stated in CPS721 syllabus CMF.

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

1 (30 points). Use PROLOG to solve the crypt-arithmetic puzzle involving multiplication and addition:

```
      S T R A W
  *         T O
    ----------
      W A R T S
 +  S T R A W
    -----------
    C H A I R S
```

Assume that each letter stands for a distinct digit and that leading digits are not zeroes. You cannot make any other assumptions.

First, solve this problem using the pure generate and test technique, without any smart interleaving, and notice how much time it takes. (Warning: it can take a few minutes or more depending on your computer.) Determine how much computer time your computation takes using the following query, where the value of `Time` will be the time taken:

```
?- Start is cputime, solve([A,C,H,I,O,R,S,T,W]), End is cputime, Time is End-Start.
```

Write your program in the provided file called **crypt1.pl** Next, solve this problem using a smart interleaving of generate and test. Write this second program in the file called `crypt2.pl` Write comments in your file which explain briefly the order of constraints you have chosen and why this has an effect on computation time. Find also how much time your program takes to compute an answer in this case. Write your session with PROLOG to the file **crypt.txt** , showing all the queries you submitted (for *both* programs) 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`. See the lecture notes for an example. To print the solution write a single rule implementing the `print_solution(ListOfVars)` predicate similar to what we considered in class.

You have to submit both your programs and your session. You will lose marks if one of the two solutions is missing. You may also lose marks if you do not put your rules/statements in the correct sections of the given files or use incorrect format.
Handing in solutions: An electronic copy of **crypt1.pl** and **crypt2.pl** and **crypt.txt** must be included in your **zip** archive.

2 (30 points). Consider scheduling a course timetable for a university student. The student wants to take a course in art, math, computer science and dance, and each of these has three hours of classes each week. Some courses offer two sections at different times.

1. There are two sections for art. One is at MWF10, and the other is at MWF11.
2. There is just one section of dance: Friday, 1–4.
3. Math is offered M11, W3, F3, or M2, W2, F11.
4. There are two sections of computer science: one on M11, W11, F12, the other on M12, W12 and W3.

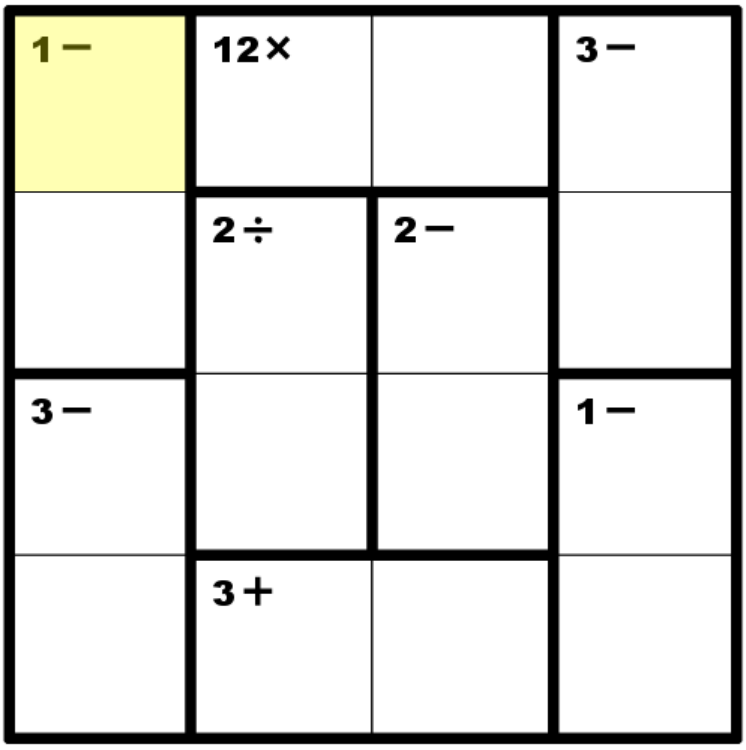
In addition, the student wants a free hour for lunch each day, either at noon or at 1pm.

Write a PROLOG program that computes a timetable for the student using methods discussed in class. Store your program in the file **timetable.pl**. Make sure that your output is easy to read: use the predicate `write(X)` and the constant `nl`. The TA should be able to query your program using `print_solution(`$List$`)` predicate that should call `solve(`$List$`)` and then print a computed solution. It is not necessary to print out a fancy formatted timetable, as long as it is obvious from your output what it is. For example, timetable we are looking for may simply say something like: section 1 of art, section 2 of math, etc. Find also how much time your program takes to compute an answer.

Recall that if the predicate that you would like to use is a part of the standard library of predicates, then you can use it in Eclipse PROLOG by default. Otherwise, if you introduce a helping predicate that is not in the library, then be sure to include the program that defines this predicate in your program file.

Handing in solutions. An electronic copy of: (a) your program in the file **timetable.pl**). You may lose marks if you do not put your rules/statements in the correct sections of the given files or use incorrect format. (b) your session with PROLOG, showing the queries you submitted and the answers returned in the file **timetable.txt**); this file should also include computation time. (c) Include both files into your ZIP file.

3 (40 points).



Many newspapers feature a puzzle called *KenKen* in the recreational section. If you never tried this puzzle before, you can read details in Wikipedia at `http://en.wikipedia.org/wiki/KenKen` Your task is to write a PROLOG program that solves the given version of a 4×4 *KenKen* puzzle. Solve this problem using smart interleaving of generate and test. The objective is to fill in the grid with the digits 1 through 4 such that:

- Each row contains exactly one of each digit
- Each column contains exactly one of each digit
- Each heavily-outlined group of cells is a cage containing digits which achieve the specified result (digits may combine in any order) using the specified arithmetical operation: addition ($+$), subtraction ($-$), multiplication ($\times$), and division ($\div$). For example, the group of numbers labeled "$12\times$" indicates that the result of multiplying the two numbers together is 12. Similarly, the group of two numbers X, Y in the top left corner labeled "$1-$" indicates that either $X - Y = 1$ or $Y - X = 1$.

- A digit can be repeated within a cage as long as it is not in the same row or column.

You can use in your program any of the predicates that we discussed in class, but you have to implement them. It will also be convenient for you to define auxiliary predicates such as $sub(X, Y, Z)$, and $dv(X, Y, Z)$, where $sub(X, Y, Z)$ holds when either $Z = X - Y$ or $Z = Y - X$, and $dv(X, Y, Z)$ holds when either $Z = X \div Y$ or $Z = Y \div X$. Implement the predicate $solve(List)$ using a single rule as in class, and incorporate all constraints in the body of your rule. The TA should be able to query your program using `print_solution(`$List$`)` predicate that should call `solve(`$List$`)` and then print a computed solution. When you print elements of $List$, make sure that the output of your program is easy to read: print your solution using the predicate `write(X)` and the constant `nl`.

Handing in solutions: (a) Add your program to the given file **kenken.pl**. You may lose marks if you do not put your rules/statements in the correct sections of the given files or use incorrect format. included in your **zip** archive; (b) Include your session with PROLOG in the file **kenken.txt**. Determine how much time it takes for your computer to solve this puzzle in this file.

4 (up to 35 points). Bonus work:

> 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. This part of the assignment is more challenging than the questions in the regular part of your assignment.* If your assignment is submitted from a group, write whether this bonus question was implemented by all people in your team (in this case bonus marks will be divided evenly between all students) or whether it was implemented by one person only (in this case only this student will get all bonus marks).

Five cows named $Flossie$, $Elsie$, $Daisy$, $Bossie$, $Maybelle$ decided to jump over the moon. They belonged to farmers $Brown$, $Jones$, $Smith$, $Nelson$, and $Ford$ (Farmers are listed independently from cows). They all jumped different heights but still fell short by some 250000 miles. They jumped (not respectively) 8 feet, 7 feet, 6 feet, 5 feet and 2 feet. All the cows were somewhat embarrassed in front of their cow friends by failing to clear the moon and came up with different excuses. Their excuses were $slipped$, $gravity$, $sore\ hoof$, $moon\ moved$, and $headache$ (not in the same order as the names of cows). Based on the clues below match the cows with their owners, the heights of their jumps, and their imaginative excuses. The contest may have been an "udder" failure, but it was a noble effort. Consider the following clues.

1. Farmer Smith's cow claimed that she did not jump over the moon because she slipped just as she jumped.
2. The cow that jumped the 3rd highest claimed she had a headache.
3. Farmer Ford, who did not own Maybelle or Elise, had a cow that only jumped 2 feet.
4. Bossie, Flossie, Elise did not claim to have headaches.
5. Flossie did not belong to farmers Smith, Nelson or Ford.
6. Maybelle and Flossie were not owned by Farmer Jones, and did not blame gravity for not jumping over the moon.
7. Farmer Nelson's cow did not claim that the moon moved.
8. The cow that claimed she slipped jumped 5 feet.
9. Farmer Brown's cow said the moon moved.
10. Farmer Nelson owns Maybelle.
11. Farmer Jones owns Daisy.
12. The cow that said that the moon moved jumped higher than the cow with a headache, which in turn jumped higher than the cow who blamed gravity.

Your task is to write a PROLOG program using the smart interleaving of generate-and-test technique explained in class to compute who owns which cow, which cow jumped which height, and which cow invented which excuses. 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. You have to write a single rule that implements the predicate **solve(**$List$**)**, as we discussed in class. You will need to be careful in your program regarding the order of constraints. Explain briefly the ordering you have chosen (write comments in your program). Remember to implement all implicitly stated and hidden constraints. They must be formulated and included in the program to find a correct solution satisfying all explicit and implicit constraints. You can use predicates from class in your program. Do not introduce any new helper predicates. Determine how much computer time your computation takes using `cputime` construct. 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. You lose marks, if the TA will see that you included some of your own reasoning into your program.

Finally, **output legibly** a solution that your program finds, so that when the TA who marks your assignment will run the query `print_solution(`$List$`)` , they will be able to read easily if your solution(s) is/are correct. Print your solution(s) on the standard output using the predicates `write(X)` and `nl`, similarly to how this is shown in slides discussed in class. You must print your solution using a rule that implements the `print_solution(List)` predicate that we considered in class. The printed output should include names of people, cows they own, the height of each jump, and the excuses. If your program finds several solutions, print all of them. Make sure this `print_solution(List)` predicate is implemented in your program, because the TA will use it as a query to test your program. You lose marks if output is not easily readable.

Copy your session with PROLOG to a file **cows.txt**, showing the queries you submitted and the answers returned (including computation time). Write a brief discussion of your program and results in this file. Put this as a comment at the top of the file.

Handing in solutions: (a) Write your program in a file called **cows.pl** that uses the same format as the others, and include it in your **zip** archive. You may lose marks if you do not use a similar format or the formatting is hard to understand (b) Add your session in PROLOG in a file called **cows.txt** that uses a similar format as the other interaction files. Determine how much time it takes for your computer to solve this bonus puzzle. Include your report into your ZIP file.

How to submit this assignment. Read regularly *Frequently Answered Questions* and replies to them that are linked from the Assignments Web 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 `crypt2.pl` or `timetable.pl` or `kenken.pl` into a 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 */opt/ECLiPSe/bin/x86_64_linux/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 on `moon`:

```
zip yourLoginName.zip crypt1.pl crypt2.pl crypt.txt timetable.pl timetable.txt
kenken.pl kenken.txt [cows.pl cows.txt]
```

where `yourLoginName` is the TMU 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 ZIP** file only (**No individual files!**)

yourLoginName.zip into the "Assignment 3" folder on D2L.

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.