cps721: Assignment 3 (100 points).
Due date: Electronic file - Wednesday, October 25, 2017, 17:00pm (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 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. By submitting this assignment you acknowledge that you read and understood the course *Policy on Collaboration* in homework assignments stated in 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 all the questions below, you should create a separate file containing rules for the `solve` 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 lose marks, if you attempt to solve a problem (or a 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.

First, you can try to solve this problem using 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, <your query>,  End is cputime, Time is End - Start.
```

Keep your program inside the comments in your file **crypt.pl** i.e., this program should be visible to a TA who will read it, but Prolog should not process this program.

Next, solve this problem using smart interleaving of generate and test. Keep this second program in the same file `crypt.pl` Write comments in your program file: 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. 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 lecture notes for an example. To print solution write a single rule implementing the `print_solution(List)` predicate similar to what we considered in class.

You have to submit your program in the file **crypt.pl** Remember to keep the first solution (with "pure generate and test") in comments in the same file **crypt.pl** You lose marks if one of the two solutions is missing.

Handing in solutions: An electronic copy of your files **crypt.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.

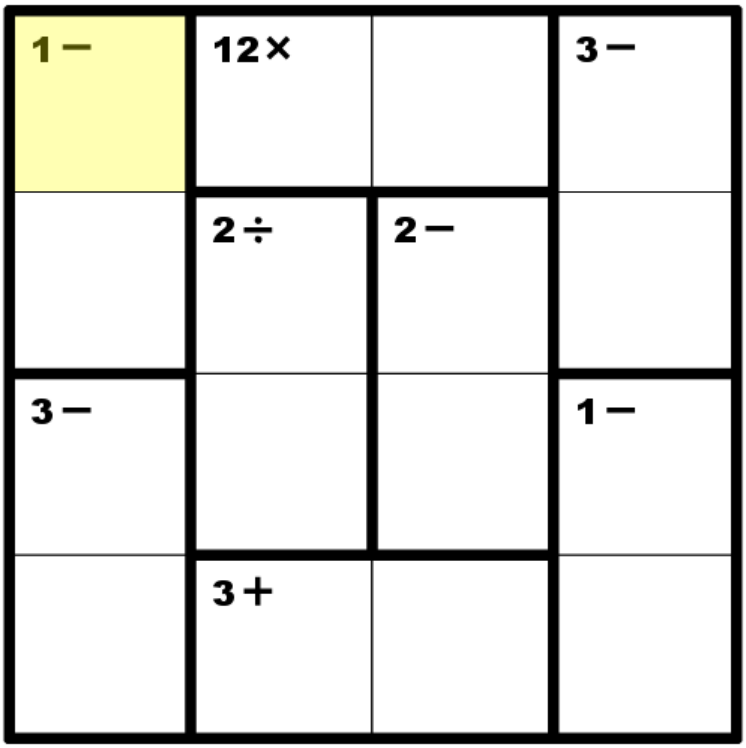
In addition, the student wants a free hour for lunch each day, either at noon or at 1pm.

Write a Prolog program that generates a timetable for the student using methods discussed in class; name you 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. The timetable I am looking for simply says 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 (the name of the file must be **timetable.pl**) (b) your session with Prolog, showing the queries you submitted and the answers returned (the name of the file must be **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 a 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 ($\div$).
- A digit can be repeated within a cage as long as it is not in the same raw 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 output of your program is easy to read: print your solution using the predicate `write(X)` and the constant `nl`.

Handing in solutions: (a) An electronic copy of the file **kenken.pl** with your Prolog program must be included in your **zip** archive; (b) Include your session with Prolog in comments on the bottom of your file **kenken.pl**. Determine how much time it takes for your computer to solve this crypt.

4. 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 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 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 Smiths 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 Nelsons cow did not claim that the moon moved.
8. The cow that claimed she slipped jumped 5 feet.
9. Farmer Brown's cow said moon moved.
10. Farmer Nelson owns Maybelle.
11. Farmer Jones owns Daisy.
12. The cow that blamed moon jumped higher than the cow with headache which in its 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: your program has to compute who owns which cow, the heights of jumps, and who 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. *Hint:* introduce a single predicate for jumps. Using this predicate, write atomic statements that

define a finite domain for all variables that you will introduce in your program. 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. Determine how much computer time your computation takes using `cputime` construct. You cannot introduce any other new predicates that have not been discussed in class. 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 embed 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(`$List$`)` , he 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 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 hight 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 legible.

Copy your session with Prolog to the 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.

Handing in solutions: (a) An electronic copy of the file **cows.pl** with your Prolog program must be included in your **zip** archive; (b) Copy your session with Prolog in the report file **cows.txt** 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 `crypt.pl` or `timetable.pl` or `kenken.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 */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 crypt.pl crypt.txt timetable.pl timetable.txt kenken.pl [cows.pl
cows.txt]
```

where `yourLoginName` is the 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.