cps721: Assignment 3 (100 points).
Due date: Electronic file - Thursday, October 25, 2018, 4:00pm (sharp).

YOU SHOULD NOT USE ";" , "!" AND "–>" IN YOUR PROLOG RULES.
You must work in groups of TWO, or THREE, you cannot work alone.
Contact the CSSU or post a message on the CSSU Facebook page to find CPS721 partners.

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 below, you should create a separate file containing rules for the `solve` predicate and any other predicates you might need. Note that your programs will have to solve each of the problems in this assignment, not you! You lose marks, if you attempt to solve a problem (or any part of the problem) yourself, and then hack a program that simply prints a solution. All work related to solving a problem should be done by your **Prolog program**, not by you.

Part 1 (35) . Use Prolog to solve the following crypt-arithmetic puzzle involving **multiplication** of SAY and MY:

```
          S A Y
    *       M Y
      ---------
        N A M E
+     A M N E
    -----------
      S T Y L E
```

Assume that each letter stands for a distinct digit and that leading digits are not zeroes.

First, try to solve this problem using *pure generate and test* technique, without any interleaving, and notice how much time it takes. Determine how much computer time your computation takes using the following query:

```
?- Start is cputime, <your query>, End is cputime, Time is End - Start.
```

The value of `Time` will be the time taken. (On my computer, this program takes less than 4 seconds using ECLiPSe Prolog release 6.) Keep this "pure generate and test" version of your program inside the comments in your file `puzzle.pl`: you lose marks if you do not provide a working version of this program. A TA will read it, but Prolog should not process this program.

Next, solve this problem using *interleaving of generate and test* approach, as discussed in class. Keep this program in the file `puzzle.pl` and make sure that the TA can compile this file, and run this program using either the main predicate **solve(List)** or **print_solution(List)** . Make sure that your output is easy to read: print your solution using the predicate `write(X)` and the constant `nl`. The predciate **print_solution(List)** must be implemented using a single rule similar to the example that we considered in class.

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, if you decide to use one. Find also how much time your program takes to compute an answer.

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

Part 2 (25). Write a Prolog program that creates a final exam schedule for a mythical university faculty. The faculty offers three programs made up from courses in the department of Computer Science, Mathematics, and Philosophy as follows:

- *Cognitive Science*: csc199, mat120, mat140;
- *Artificial Intelligence*: csc199, csc108, csc148, mat140;
- *Philosophy of Mind*: csc199, mat101, phl250.

The final exams for all courses are to take place between December 11 and December 14 inclusive, subject to the following constraints:

(a) No program should have two exams on the same day.

(b) phl250 must be scheduled on the first day; mat140 cannot be scheduled on the last day.

(c) Within a Department, all exams should appear in order, starting with the lower numbered courses. For example, in Mathematics, the exam for mat120 should not be before mat101, nor be after mat140.

Your program should choose exam days for all the courses. As in the first question, you will need to be careful in your program regarding the order of constraints. Explain briefly the ordering you have chosen (write comments in you program). Make sure you print clearly the solutions: you lose marks if output is difficult to read.
Handing in solutions: (a) An electronic copy of your file **schedule.pl** must be included in your **zip** archive; (b) Your zip file must also contain your session with Prolog, showing the query `solve(List)` you submitted and the answers returned (the name of the file must be **schedule.txt**); this file should also include computation time. Make sure that your answers are easy to read.

Part 3 (40). In the town Mooseville, there are just six computer systems connected to the Internet. (Of course, each system may have many users on it.) The six computer systems exchange files with the systems linked with them on the map below, and system 1 on the map also exchanges files with systems outside of Mooseville, providing the town with its connection to the rest of the world. Thus, electronic mail travels from system to system along the links shown on the map:

```
(6) -- (5) -- (3) -- (2) -- (1) --> (Rest of the world)
               |
              (4)
```

```
Note: 4 has a link to 3, 5 has a link to 3, 5 does not have a direct link to 4.
The path from 6 to "rest of the world" is: 6, 5, 3, 2, 1, out.
```

Each of the computer systems has a different Internet address (`bananas.com`, `firstbank.com`, `netvue.com`, `pricenet.com`, `sysworld.com`, and `univmoose.edu`), and a different man or woman as its systems administrator. The six systems administrators include three women named Catarina, Lizzie, and Mona, and three men named Anthony, Daniel, and Jaime; last names are (not respectively) Elby, Kim, Osborne, Tsuji, Wolverton, and Zickerman. Your Prolog program must match each computer system on the map with its Internet address and with the full name of its system administrator. Suppose the following facts are true.

1. Email between Lizzie and Osborne must also pass through the system with the Internet address `pricenet.com` (and possibly other systems as well).
2. Mona's and Wolverton's systems exchange files directly; one of these systems' addresses is `netvue.com`
3. Email between Anthony and Jaime must also pass through Elby's system (and possibly others).
4. Among the systems in town, Daniel's exchanges files directly only with the one whose address is `sysworld.com`
5. Email between Jaime and Ms. Tsuji must also pass through the system whose address is `univmoose.edu`

6. The system whose address is `bananas.com` has a female systems administrator.
7. Email between Kim and the rest of the world must also pass through the system whose address is `firstbank.com`
8. Among the systems in town, Zickerman's exchanges files directly only with Catarina's and the system whose address is `netvue.com`. Zickerman's system does not have direct links with any other computer system in the town.

Assume that in the constraints (1), (3), (5), (7) the packages can possibly pass through more than one system (not only one computer system that is mentioned explicitly). Write a Prolog program **network.pl** that solves this problem using interleaving of generate and test technique considered 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 you program file). Make sure the answers returned by your program are easy to read: you lose marks if output is obscure.

Hint: you might wish to consider the 3-argument predicate *trip(Origin,Destination,Path)* as a helping predicate and use it in your program. This predicate is true, if *Path* is the list of computer systems connecting *Origin* with *Destination* (*Path* includes computer systems *Origin* and *Destination* if they are different from each other). If *Origin=Destination*, then *Path* has only one element. There is a direct link on the map between consecutive elements of *Path*. Using this predicate *trip*, you can formulate the constraint that email from *Origin* to *Destination* must pass through a computer system *Intermediate* (in other words, that the computer system *Intermediate* is located between *Origin* and *Destination* in the computer network on the map).

Handing in solutions: your zip file must contain (a) An electronic copy of your program (**network.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 a copy inside comments in the same file **network.pl**).

4. Bonus work (up to 50):

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.* 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).

John liked Asian food so much that he ate it five days a week. However, John only liked five Asian dishes. Namely, he liked *egg rolls, chow mein, sour soup, fried rice* and *Peking duck*. John always ate two courses every meal. To keep his meals interesting, he mixed the dishes up so that he never ate the same combination twice in one week. John did not have this problem with his drink order as he drank something different with every meal. He liked to drink *tea, coffee, milk, juice and water*. Write a Prolog program based on the following clues.

1. John never ate the same combination of dishes twice in one week.
2. The only dish John ate two days in a row was *Peking duck* on Tuesday and Wednsday.
3. John didn't order *sour soup* on Tuesday, Wednesday or Thursday.
4. John's fist course on Tuesday was *chow mein*, and he had *chow mein* again for second course the day after his 1st course was *duck*.
5. John ate no *egg rolls* on Monday or Thrusday, and he drank *coffee* after he drank *milk*, and *tea* the day after he drank *coffee*.
6. John ordered *egg rolls* at least twice in a week.
7. *Sour soup* can be ordered only as the first dish, but not as the second.
8. John drank *water* afer he drank *tea*, and he drank *juice* on Friday.

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 the full meal on each weekday including two courses and a drink. 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. *Hints:* introduce a predicate for weekdays. Using this first predicate, write atomic statements that define a finite domain for all variables related to drinks. Additionally, introduce another predicate for dishes and use it to write atomic statements. The variables for the first and second courses per weekday should take values from the domain of dishes.

You have 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. 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`. You must print solution using a rule that implements the `print_solution(List)` predicate that we considered in class. If your program finds several solutions, print all of them. You lose marks if output is not legible.

Copy your session with Prolog to the file **food.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 **food.pl** with your Prolog program must be included in your **zip** archive; (b) Copy your session with Prolog in the report file **food.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. Do not bring any printouts to class. 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 `puzzle.pl` or `schedule.pl` or `network.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 on `moon`:

```
zip yourLoginName.zip puzzle.pl puzzle.txt schedule.pl schedule.txt network.pl
```

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.