cps721: Assignment 3 (100 points).
Due date: Electronic file - Monday, October 26, 2015, before 21:00.

Hand in a printed copy at start of class on Tueday, Oct 27.

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 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 (35 points). Use Prolog to solve the following crypt-arithmetic puzzle involving addition:

```
          C R O S S
    +     R O A D S
      -----------
      D A N G E R
```

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. The main predicate should be called **solve1**. (Warning: it can take a minute or more depending on your computer.) Determine how much computer time your computation takes using the following query:

```
?- Start is cputime, <your query>, Finish is cputime, T is Finish - Start.
```

The value of the variable `T` will be the time taken. Keep this program with the predicate **solve1** in your file `puzzle.pl`

Next, solve this problem using smart interleaving of generate and test, as discussed in class. This second program should have the main predicate **solve2**. Keep this program in the same file `puzzle.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. 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. Provide information about your hardware: CPU and memory installed on your computer.

For both programs, save your session with Prolog into 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 on the standard output using the predicate `write(X)` and the constant `nl`. You can print solution using a Prolog rule similar to the `print_solution` predicate that we considered in class.

Handing in solutions: (a) An electronic copy of your files **puzzle.pl** and **puzzle.txt** must be included in your **zip** archive; (b) Hand in printouts of these 2 files in class.

2 (25 points).

Luigi's Logical Pizzeria served five different sized pizzas: small, medium, large, extra-large and gigantic. They featured four different toppings, namely mushroom, sausage, pepperoni, onion, or plain cheese pizza (toppings are listed independently from sizes). One night, five people named Ben, Cindy, Emily, Mary, Rob (sorted alphabetically), called to order pizzas for takeout. Each ordered a different size and a different topping or plain. It is known that all people called at different time for their pizza, i.e., someone called 1st, someone else called 2nd, 3rd, 4th, 5th, but it is not know who called when. Based on the clues provided below, match the callers with the size of pizza they ordered, the toppings they wanted, and the order in which they called.

1. Rob did not call 1st or 5th as those callers ordered either a small or a gigantic pizza with either sausage or mushrooms on them, and Rob did not like sausage or mushrooms.

2. Mary called before Cindy, and Ben called after Rob, but before Mary.

3. The extra-large was ordered after the large and neither of those pizzas had onions on them. The medium pizza was ordered immediately after the gigantic mushroom pizza was ordered.

4. Ben ordered a pizza with a topping on it.

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 ordered what and when. 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**, as we discussed in class. *Hint:* introduce a single predicate call(X) representing the call number. 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 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 is correct. Print your solution 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, order number, size of pizza each person has ordered, and the topping. 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 **logic.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 your file **logic.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 (the name of the file must be **logic.txt**); this file should also include computation time. Make sure that the solution your program prints is easy to understand. (c) Hand in a printout of both files **logic.pl** and **logic.txt** in class.

3. (40 points).

Consider the crossword puzzle shown in Figure below. Using smart interleaving of generate and test technique, write a Prolog program that finds six 3-letter words: three words read across (rows $R1, R2, R3$) and three words read down (columns $C1, C2, C3$). Note that the program has to find six distinct words. Each word must be chosen from the list of several 3-letter English words (a file is posted in the Assignments folder on my.ryerson.ca). Include these words as atomic statements in your program. Below you have to consider two ways to represent the crossword puzzle shown in the figure as a CSP.

```
 _____ _____ _____
|R1,C1| C2  |  C3 |
|     |     |     |
|_____|_____|_____|   Download word list from CPS721 shell on my.ryerson.ca
|R2   |     |     |
|     |     |     |
|_____|_____|_____|
|R3   |     |     |   Figure: A crossword puzzle to be solved with
|     |     |     |             6 distinct words (3 letters each).
|_____|_____|_____|
```

The first approach is to represent the word positions $\{R1, R2, R3, C1, C2, C3\}$ as variables, with the set of words as possible values. The constraints are that where the words intersect, the letter is the same. Write a Prolog program that implements this approach. Call your main predicate **solve1(List)**. Save your program in the file **crossword.pl**

The second way is to represent the nine squares as variables. The domain of each variable is the set of letters of the English alphabet. The constraints are that there is a word (in the provided set of words) that contains the corresponding letters. For example, the top-left square and the center-top square cannot both have the value z, because there is no word starting with zz. Write another Prolog program that implements this second approach. Be careful with choosing the order of generating values and testing constraints. Call your main predicate **solve2(List)** and keep it in the same file **crossword.pl**

Both your programs have to print clearly the 6 distinct words that solve the given crossword puzzle. Test both of your programs: save queries and answers in the file **crossword.txt** Find how much time takes solving the puzzle for both of your programs. Write a brief report. Which representation leads to a more efficient solution? Give the evidence for your answer. Explain the reasons: you can give arguments based on combinatorics of search that both programs do.
Handing in solutions: (a) An electronic copy of the file **crossword.pl** with your Prolog program must be included in your **zip** archive; (b) Include your report, the file **crossword.txt** , in your **zip** archive; (c) Hand in printouts of **crossword.pl** and **crossword.txt** in class.

How to submit this assignment. Hand in printouts of all files before beginning of class. On the first page of your printout **write the name of your electronic ZIP file** to make sure the TA can locate it easily among other ZIP files. **Staple all pages!** 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 `puzzle.pl` or `logic.pl` or `crossword.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 any *moon* Linux server:

```
zip yourLoginName.zip puzzle.pl puzzle.txt logic.pl logic.txt crossword.pl crossword.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 numbers* and *names* of all people who participated in discussions (see the course management form). You may be penalized for not doing so. Second, on one of the `moons` run the command

```
submit-cps721  yourLoginName.zip
```

If you need more information about **zip**, read manual pages on *moon*. Make sure you have **ssh** software on your home computer. You might need it to establish a remote connection with departmental servers, if you are going to submit your assignment from home. Otherwise, submit your assignment from labs at any time. 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. Make sure that your Computer Science account at Ryerson is properly configured and you can easily access it from Labs (or from home, if you wish). Talk to one of the system administrators in ENG246 or ENG248 *in advance* if you experience any difficulties with your login name, password, or with your computer account.

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 in time.