COL226: Programming Languages
II semester 2022-23

Assignment: Four Simple Prolog Problems

---

1. Write a predicate `subsequence(S,L)` such that `S` is a list of elements occurring in that order, but not necessarily successively, in list `L`.

   For example, `subsequence([c,d,e],[a,b,c,d,e,f])` and `subsequence([c,e],[a,b,c,d,e,f])` should evaluate to true, whereas `subsequence([f,a],[a,b,c,d,e,f])` should be false.

   Call your main predicate `subsequence(...)`.

2. Write a predicate `has_no_triplicates(L)` that evaluates to true if the list of elements `L` does not contain three (or more) copies of an element, and false otherwise.

   Call your main predicate `has_no_triplicates(...)`.

3. Given a list of integers, find a correct way of inserting the arithmetic operator symbols `+`, `-`, and `=`, such that the result is a correct equation. For example, with the list of numbers [2,3,4,6,9] we can form the equations 2=3-4-6+9, 2-3+4+6=9 and -2+3-4=6-9 (and perphaps more!).

   Call your main predicate `arith(...)`.

4. Alice, Bob, Carol, and Davis had to cross a river using a canoe that held only two persons. In each of the three crossings from the left to the right bank, the canoe had two persons, and in each of the two crossings from the right to the left bank, the canoe had one person. Alice was unable to paddle when someone else was in the canoe with her. Bob was unable to paddle when anyone else but Carol was in the canoe with him. Each person paddles for at least one crossing. Who paddled twice?

   Write a Prolog program that solves the problem above. As part of the solution it should print out all the crossings, with the paddler listed first.

   Call your main predicate `abcd(...)`.

What you need to do

Create Prolog files named respectively

1. `subseq.P` with main clause called `subsequence(...)`
2. `triplicates.P` with main clause called `has_no_triplicates(...)`
3. `arith.P` with main clause called `arith(...)`
4. `ABCD.P` with main clause called `abcd(...)`

which are Prolog clause lists along with appropriate queries that solve the above problems and zip them up into a file `ass5.zip` and submit it on Gradescope.

Note for all assignments in general

1. Some instructions here may be overridden explicitly in the assignment for specific assignments
2. Upload and submit a single zip file called `<entryno>.zip` where `entryno` is your entry number.
3. You are *not* allowed to change any of the names or types given in the specification/signature. You are not even allowed to change upper-case letters to lower-case letters or vice-versa.
4. The evaluator may use automatic scripts to evaluate the assignments (especially when the number of submissions is large) and penalise you for deviating from the instructions.
5. You may define any new auxiliary functions/predicates if you like in your code besides those mentioned in the specification.
6. Your program should implement the given specifications/signature.
7. You need to think of the *most efficient way* of implementing the various functions given in the specification/signature so that the function results satisfy their definitions and properties.
8. In a large class or in a large assignment, it is not always possible to specify every single design detail and clear each and every doubt. So you are encouraged to also include a `README.txt` or `README.md` file containing
   - all the decisions (they could be design decisions or resolution of ambiguities present in the assignment) that you have taken in order to solve the problem. Whether your decision is "reasonable" will be evaluated by the evaluator of the assignment.
   - a description of which code you have "borrowed" from various sources, along with the identity of the source.
   - all sources that you have consulted in solving the assignment.
   - name changes or signature changes if any, along with a full justification of why that was necessary.
9. The evaluator may look at your source code before evaluating it, you must explain your algorithms in the form of comments, so that the evaluator can understand what you have implemented.
10. Do *not* add any more decorations or functions or user-interfaces in order to impress the evaluator of the program. Nobody is going to be impressed by it.
11. There is a serious penalty for code similarity (similarity goes much deeper than variable names, indentation and line numbering). If it is felt that there is too much similarity in the code between any two persons, then both are going to be penalized equally. So please set permissions on your directories, so that others have no access to your programs.
12. To reduce penalties, a clear section called "**Acknowledgements**" giving a detailed list of what you copied from where or whom may be be included in the `README.txt` or `README.md` file.