Project 4

Due: Apr 13, before midnight.

Important Reminder: As per the course Academic Honesty Statement, cheating of any kind will minimally result in your letter grade for the entire course being reduced by one level.

This document first provides the aims of this project. It then lists the requirements as explicitly as possible. It then hints at how these requirements can be met. Finally, it describes how it can be submitted.

Aims

The aims of this project are as follows:

- To give you more exposure to recursive programming.
- To expose you to Prolog programming.

Requirements

Implement all the procedures specified in prj4_sol.pro using swipl Prolog. You may define auxiliary procedures as needed.

The submitted file should have all tests unblocked.

An example LOG file gives examples of the use of these procedures as do the tests provided in the skeleton file prj4_sol.pro.

You are not allowed to use any non-logical features of Prolog other than arithmetic procedures like `is/2`, `=:=/2` or `=</2`. Non-logical features include the explicit cut `!`, implicit cut within `->`, `assert`, `retract`, `record`, etc.

Provided Files

You should use the provided prj4-sol directory as a starting point for your project by copying it into your `i571?/submit` directory. It contains the following files:

prj4_sol.pro
: A skeleton file which contains the specifications for the procedures you are required to write as well as tests for those procedures.

  It is worth noting that the provided test code is several tens of times longer than the implementation code. Specifically, none of the procedures you are required to implement need more than around 10 lines of code.

README
: A README file which must be submitted along with your project. It contains an initial header which you must complete (replace the dummy entries with your name, B-number and email address at which you would like to receive project-related email). After the header you may include any content which you would like read during the grading of your project.

The extras directory contains the following files:

LOG

A sample log file.

Tests

The prj4_sol.pro skeleton file contains unit tests for the procedures you are required to implement. Note that specifying a test as `nondet` means that the test may succeed with more possible answers pending. The `set` specification specifies a list of all possible answers after backtracking through the test.

You can run the tests using Prolog's `run_tests/0` or `run_tests/1`.

```
$ prolog
...
%load file
?- ['prj4_sol.pro'].
true.

% run all tests using provided skeleton file
% where all tests are blocked
?- run_tests.
% PL-Unit: sublist_lengths blocked: TODO
% PL-Unit: same_length_sublists blocked: TODO
% PL-Unit: fibonacci_sublists blocked: TODO
% PL-Unit: assoc_lookup blocked: TODO
% PL-Unit: assoc_replace blocked: TODO
% PL-Unit: add_to_plus_expr blocked: TODO
% PL-Unit: named_to_op_expr blocked: TODO
% PL-Unit: named_expr_eval blocked: TODO
% PL-Unit: named_expr_to_prefix_tokens blocked: TODO
% PL-Unit: op_expr_to_prefix_tokens blocked: TODO
% No tests to run
true.

% ... develop ...

% run all tests after completing project
?- run_tests.
% PL-Unit: sublist_lengths ....  passed 0.000 sec
% PL-Unit: same_length_sublists ....... ...
% PL-Unit: fibonacci_sublists ............  ...
% PL-Unit: assoc_lookup ........ passed 0.000 sec
% PL-Unit: assoc_replace ..... passed 0.000 sec
% PL-Unit: add_to_plus_expr ............ passed 0.001 sec
% PL-Unit: named_to_op_expr ............. ...
% PL-Unit: named_expr_eval ............... passed 0.001 sec
% PL-Unit: named_expr_to_prefix_tokens .... ...
% PL-Unit: op_expr_to_prefix_tokens ....... ...
% All 123 tests passed
true.

% run tests only for a specific procedure
?- run_tests(op_expr_to_prefix_tokens).
% PL-Unit: op_expr_to_prefix_tokens ........ ...
% All 15 tests passed
true.

% terminate prolog
?- halt.
$
```

You can also run the tests directly from the Unix command line: by running your implementation file as a script:

```
# ensure in project directory
$ cd ~/i571?/submit/prj4-sol

# run all tests
$ ./prj4_sol.pro
% PL-Unit: sublist_lengths .... passed 0.000 sec
% PL-Unit: same_length_sublists ... ...
...
% PL-Unit: op_expr_to_prefix_tokens .... ...
% All 123 tests passed

# run selected tests
$ ./prj4_sol.pro op_expr_to_prefix_tokens
% PL-Unit: op_expr_to_prefix_tokens ..... ...
% All 15 tests passed
$
```

Hints

The following points are worth noting:

- Review the class Prolog slides. In particular, review the slides on Prolog Programming Heuristics.
- If you do not understand the specs for a procedure, please look at the tests or LOG for examples of the expected use of the procedure.
- Each procedure **`proc`** that you are required to write has a skeleton rule of the form:

  ```
  proc(_Arg1Name, ...) :- 'TODO'.
  ```

  Replace this skeleton line with facts and rules for the procedure.
- The tests provided in the skeleton file are currently blocked. To run the tests for an individual procedure, unblock the tests by removing the **`blocked('TODO')`** argument.
- Since there are no types or declarations in Prolog, it is very easy to have a bug resulting from a typo in a variable name. Prolog tries to prevent this by generating warnings about **singleton variables**; i.e, any variable which occurs only once within a Prolog rule. For example, the rule

  ```
  member(X, [Element|Elements]) :-
    member(X, Elements).
  ```

  would produce a warning that **`Element`** is a singleton variable.

  If the singleton variable is intentional, then the warning can be avoided by replacing the name with one starting with an underscore **_**, as in the code shown below:

  ```
  member(X, [_Element|Elements]) :-
    member(X, Elements).
  ```
- Prolog has a pretty powerful debugging model. You can turn on tracing using **`trace/0`** or **`trace/1`** and turn it off using **`nodebug/0`**, spying on a particular procedure using **`spy/1`** and turn off using **`nospy/1`** of **`nospyall`**. If running within a graphical environment, you can turn on GUI debugging using **`gtrace`**.

Submission

Before submitting your project, update your README to specify the status of your project. Document any known issues.

Submit using a procedure similar to that used in your previous project. [As usual, gradescope will be set up early next week].