Homework: Logic Programming

Learning Objectives:

1. Problem-solving using logic programming paradigm
2. PROLOG programming

Instructions:

- Total points 50 pt
- Early deadline: April 12 (Wed) at 11:59 PM; Regular deadline: April 14 (Fri) at 11:59 PM (or till TAs start grading the homework)
- Download and install Swi-PROLOG `http://www.swi-prolog.org/`.
- You can use any code in the lecture notes directly, but you are responsible for correctly using them.
- Please zip **".pl" files and output files (e.g., screenshot) for all the solutions** and submit it to Canvas.

Questions:

1. (8 pt) [PROLOG for numbers]

   (a) (4 pt) Write a PROLOG program to compute the sequence of Lucas numbers given its length.

   Example:

```
?- lucas(0,X).
X = [].
?- lucas(1,X).
X = [2].
?- lucas(3,X).
X = [2, 1 ,3].
?- lucas(6,X).
X = [2, 1, 3, 4, 7, 11].
```

   Solution:

```
reverse([], Z, Z).
reverse([H|T], Z, A):-
  reverse(T, Z, [H|A]).

luc(0,[2]).
luc(1,[1, 2]).
luc(N, [R, X, Y|Zs]):-
  N>1,
  N1 is N-1,
  luc(N1, [X,Y|Zs]),
```

```
  R is X + Y.

lucas(0, []).
lucas(N, Z):-
  N1 is N-1,
  luc(N1, Zr),
  reverse(Zr, Z).
```

(b) (4 pt) Write a PROLOG program to compute LCM of two numbers.

Example:

```
?- lcm(10,6,X).
X = 30.
?- lcm(10,5,X).
X = 10.
```

Solution:

```
lcm(X, Y, Z):- gcd(X, Y, Y1), Z is X*Y/Y1.
gcd(0, A, A):- A > 0, !.
gcd(A, B, Z):- A >= B, X is A-B, gcd(X,B,Z).
gcd(A, B, Z):- A < B, X is B-A, gcd(X,A,Z).
```

2. (15 pt) [PROLOG for list]

(a) (5 pt) Write a PROLOG program to subtract two lists.

Example:

```
?- sublist([], [],Z).
Z = [].
?- sublist([1,1], [1,2,3],Z).
Z = [0, -1, 3]
?- sublist([1,3,2,5], [1,2,3],Z).
Z = [0, 1, -1, 5]
```

Solution:

```
append([], L, L).
append([X|Xs], L, [X|Ys]):-
  append(Xs, L, Ys).

sublist([],[],[]).
sublist([], L, Z):-
 append([], L, Z).
sublist(L, [], Z):-
 append([], L, Z).
sublist([X|A], [Y|B], Z):-
  V is X-Y,
  sublist(A, B, W),
  append([V], W, Z).
```

(b) (5 pt) Write a PROLOG program to duplicate the elements of a list a given number of times.

Example:

```
?- duplicate ([],3,X).
X = [].
```

```
?- duplicate ([a,b,c],2,X).
X = [a,a,b,b,c,c].
?- duplicate ([a,b,c],3,X).
X = [a,a,a,b,b,b,c,c,c].
```

Solution:

```
append([], L, L).
append([X|Xs], L, [X|Ys]):-
  append(Xs, L, Ys).

duplicate([], _, []).
duplicate([X], 1, [X]).
duplicate([X], M, [X|L]) :-
    N is M - 1,
    duplicate([X], N, L).
duplicate([X|Xs], M, T) :-
    duplicate([X], M, Z),
    duplicate(Xs, M, V),
    append(Z, V, T).
```

(c) (5 pt) Write a PROLOG program to find the immediate left neighbor in a given list and number.

Example:

```
?- neighbor(1, [2,3,1,4],X).
[3]
?- neighbor(2, [2,3,1,4],X).
[]
?- neighbor(5, [2,3,1,4], X).
[]
```

Solution:

```
neighbor(_,[_|[]],[]).
neighbor(X,[Y, X|_],[Y]).
neighbor(X,[_, B|Tail],Z):-
  neighbor(X,[B|Tail],Z).
```

3. (5 pt) [PROLOG for integer constraints] Write a PROLOG program to generate the integer values of x, y and z that can satisfy the constraints in the following C program. If no such values can be found, return `false`.

```
if (x^2 == y-2) {
  if (y == x + 8) {
    y = x + 8
    z = x + y - 5;
  }
  else {
    x = y + 4
    z = x - y + 5;
  }
}
```

Example:

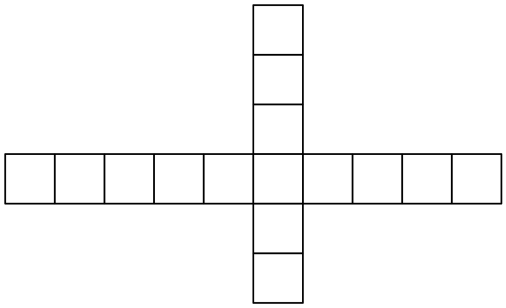
```
?- genconstraint(-2, 6, -1).
true
?-genconstraint(2, 4, 6).
false
?-genconstraint(3, 11, Z).
Z = 9
?-genconstraint(3, 10, Z).
false
```

Solution:

```
genconstraint(X,Y,Z):-((X*X) =:= Y-2) -> ((Y =:= (X+8)) ->  Y is X+8, Z is X+Y
    -5; X is Y+4, Z is X-Y+5).
```

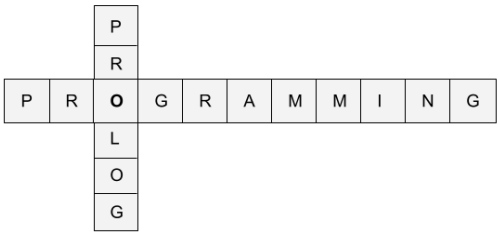
4. (6 pt) [PROLOG for word puzzle] Write a PROLOG program to solve a word puzzle: find the two words from the following poem to fill the word puzzle shown in the following figure. (Note that: directly providing the solution for the puzzle without providing the PROLOG program does not get credit.)

Poem: Snowflakes dance in the air. A winter ballet without a care.



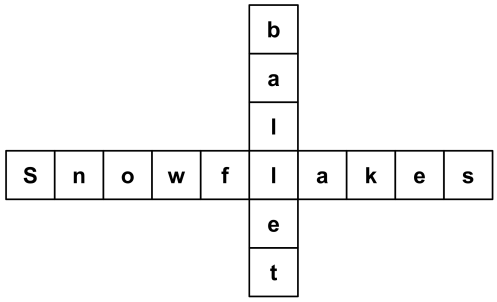
Here is an example word puzzle for your reference.

Poem: PROLOG programming rocks! See below the solution.



Solution:

```
puzzle(X,Y):-
  word(X,[_,_,_,_,_,_,Z,_,_,_,_,_]),
  word(Y,[_,_,_,_,Z,_,_]).
```




```
word(snowflakes,[s,n,o,w,f,l,a,k,e,s]).
word(dance,[d,a,n,c,e]).
word(in,[i,n]).
word(the, [t,h,e]).
word(air, [a,i,r]).
word(A, [A]).
word(winter, [w,i,n,t,e,r]).
word(ballet, [b,a,l,l,e,t]).
word(without, [w,i,t,h,o,u,t]).
word(a, [a]).
word(care, [c,a,r,e]).
```

---

5. (6 pt) [PROLOG for logic puzzle] Write a PROLOG program (including the query) to solve the following logic puzzle:

   Five people were eating apples, A finished before C, but behind B. E finished before D, but behind C. What was the finishing order?

   Solution:

```
puzzle(Hs):-
 length(Hs, 5),
 member(a, Hs),
 member(b, Hs),
 member(c, Hs),
 member(d, Hs),
 member(e, Hs),
 order(a, c, Hs),
 order(b,a, Hs),
 order(e,d, Hs),
 order(c,d, Hs).

order(a, c, Ls):-append(_,[a,c|_], Ls).
order(b, a, Ls):-append(_,[b,a|_], Ls).
order(e, d, Ls):-append(_,[e,d|_], Ls).
order(c, d, Ls):-append(_,[c,d|_], Ls).
```

   Query: puzzle(Hs).

   Finishing order: Hs = [b, a, c, e, d]

6. (10 pt) [PROLOG for parsing]

   (a) (7 pt) Write a PROLOG parser for the grammar in HW1 Q1.

   Example:

```
?- parse([]).
true.
?- parse([if, '(', a, <, b, ')', then, '{', b, +, a, '}']).
true.
?- parse([if, '(', a, >, b, ')', then, '{', b, +, a, '}', else, '{', b, =,
    0, '}']).
true.
parse([if, '(', a, <, b, ')', then, '{', [if, '(', a, <, 0, ')', then, '{',
    [], '}'], '}']).
true.
parse([if, '(', a, >, b, ')', then, '{', [], '}', else, '{', [if, '(', a, <,
     0, ')', then, '{', [], '}'], '}']).
true.
?- parse([if, '(', a, =, b, ')', then, '{', a, +, b, '}']).
false.
?- parse([if, '(', a, >, b, ')', then, '{', a, -, x, '}']).
false.
?- parse([if, '(', a, >, b, ')', then, a, -, b]).
false.
?- parse([if, '{', a, >, b, '}', then, '{', a, -, b, '}']).
false.
```

Solution:

```
parse([]).
% F -> if B then { S }; where S -> TNT
parse([E1, E2, E3, E4, E5, E6, E7, E8, E9, E10, E11, E12]):-
  if(E1), cond(E2, E3, E4, E5, E6),
  then(E7), sqBrackOpen(E8), stmt(E9, E10, E11), sqBrackClose(E12).
% F -> if B then { S } else { S }; where S -> TNT
parse([E1, E2, E3, E4, E5, E6, E7, E8, E9, E10, E11, E12, E13, E14, E15, E16
    , E17, E18]):-
  if(E1), cond(E2, E3, E4, E5, E6),
  then(E7), sqBrackOpen(E8), stmt(E9, E10, E11), sqBrackClose(E12),
  else(E13), sqBrackOpen(E14), stmt(E15, E16, E17), sqBrackClose(E18).
% F -> if B then { S }; where S -> F
parse([E1, E2, E3, E4, E5, E6, E7, E8, E9, E10]):-
  if(E1), cond(E2, E3, E4, E5, E6),
  then(E7), sqBrackOpen(E8), stmt(E9), sqBrackClose(E10).
% F -> if B then { S1 } else { S }; where S1 -> F and S -> TNT
parse([E1, E2, E3, E4, E5, E6, E7, E8, E9, E10, E11, E12, E13, E14, E15, E16
    ]):-
  if(E1), cond(E2, E3, E4, E5, E6),
  then(E7), sqBrackOpen(E8), stmt(E9), sqBrackClose(E10),
  else(E11), sqBrackOpen(E12), stmt(E13, E14, E15), sqBrackClose(E16).
% F -> if B then { S } else { S1 }; where S1 -> F and S -> TNT
parse([E1, E2, E3, E4, E5, E6, E7, E8, E9, E10, E11, E12, E13, E14, E15, E16
    ]):-
  if(E1), cond(E2, E3, E4, E5, E6),
  then(E7), sqBrackOpen(E8), stmt(E9, E10, E11), sqBrackClose(E12),
  else(E13), sqBrackOpen(E14), stmt(E15), sqBrackClose(E16).
% F -> if B then { S } else { S }; where S -> F
parse([E1, E2, E3, E4, E5, E6, E7, E8, E9, E10, E11, E12, E13, E14]):-
  if(E1), cond(E2, E3, E4, E5, E6),
  then(E7), sqBrackOpen(E8), stmt(E9), sqBrackClose(E10),
  else(E11), sqBrackOpen(E12), stmt(E13), sqBrackClose(E14).
% B -> (T E T)
cond(E1, E2, E3, E4, E5):-
```

```
  condBrackOpen(E1), t(E2), e(E3), t(E4), condBrackClose(E5).
% S -> F | TNT
stmt(E1):- parse(E1).
stmt(E1, E2, E3):-
  t(E1), n(E2), t(E3).
% T -> a | b | 0
t(a).
t(b).
t(0).
% E -> > | <
e(>).
e(<).
% N -> + | * | =
n(+).
n(*).
n(=).
if(if).
then(then).
else(else).
condBrackOpen('(').
condBrackClose(')').
sqBrackOpen('{').
sqBrackClose('}').
```

(b) (3 pt) Write a query to generate 3 examples of such language.

Solution:

For this implementation, just type `parse (A).` and continue to ask for 2 more examples by pressing *TAB*. If the answer is truncated, try pressing `w`.

```
?- parse(A).
A = [if, '(', a, >, a, ')', then, '{', a, +, a, '}']
A = [if, '(', a, >, a, ')', then, '{', a, +, b, '}']
A = [if, '(', a, >, a, ')', then, '{', a, +, 0, '}']
A = [if, '(', a, >, a, ')', then, '{', a, *, a, '}']
A = [if, '(', a, >, a, ')', then, '{', a, *, b, '}']
A = [if, '(', a, >, a, ')', then, '{', a, *, 0, '}']
```