

All exercise numbers refer to the Sterling and Shapiro text book.

1. Exercises 8.3.1 (i), (iii), (vi), (vii) {SAME FROM PREVIOUS ASSIGNMENT}*

(i) Write an iterative version for *triangle(N,T)*, posed as Exercise 8.2(i).

Ans.

% triangle(N,T): T is sum of first N natural numbers.

triangle(N,T) :- triangle(0,N,0,T).

triangle(I,N,A,T) :-

I < N, I1 is I+1,

A1 is A+I1, triangle(I1,N,A1,T).

triangle(N,N,T,T).

```
?- ['a02q04.pl'].
true.

?- triangle(3,T).
T = 6 ;
false.

?- triangle(5,T).
T = 15 ;
false.

?- 
```

(iii) Rewrite Program 8.5 so that the successive integers are generated in descending order.

Ans.

% between(I,J,K): K is an integer between the integers I and J inclusive.

between(I,J,I) :- I <= J.

between(I,J,K) :- I < J, I1 is I+1, between(I1,J,K).

Program 8.5 Generating a range of integers

Ans.

% between(I,J,K): K is an integer between the
% integers I and J inclusive.

between(I,J,J) :- I <= J.

between(I,J,K) :- I < J, J1 is J-1, between(I,J1,K).

```
?- between(1,7,X).
X = 7 ;
X = 6 ;
X = 5 ;
X = 4 ;
X = 3 ;
X = 2 ;
X = 1 ;
false.

?- 
```

(vi) Write a program to find the minimum of a list of integers.

Ans.

% min(List,least): least integer from List

%min([M|_],M) :- min([M|_],M).

min([M],M).

min([H|T],H) :-

min(T,M), H <= M.

min([H|T],M) :-

min(T,M), M < H.

```
?- ['a02q04.pl'].
true.

?- min([1,2,3,0,12],M).
M = 0 ;
false.

?- min([21,42,-56,23],M).
M = -56 ;
false.

?- 
```

(vii) Rewrite Program 8.11 for finding the length of a list so that it is iterative. (Hint: Use a counter, as in Program 8.3)

Ans.

% length_itr(List,A,L): L is length of List,

% A being accumulator

lengthL([],0).

lengthL([H|T],L) :- length_itr([H|T],0,L).

length_itr([],A,A).

length_itr([H|T],A,L) :-

A1 is A+1, length_itr(T,A1,L).

```
?- ['a02q04.pl'].
true.

?- lengthL([1,2,3,0,12],L).
L = 5.

?- lengthL([a,b,c,d,e,f],L).
L = 6.

?- 
```

2. Exercises 9.2.1 (i), (ii), (iv), (v).

(i) Define a predicate *occurrences*(*Sub*,*Term*,*N*), true if *N* is the number of occurrences of subterm *Sub* in *Term*. Assume that *Term* is ground.

Ans.

```
occurrences(Sub,Term,N) :-
    occurrences(Sub,Term,0,N).

occurrences(Term,Term,N,R) :-
    N1 is N+1,
    compound(Term), !,
    Term=..[_|Args],
    occurrencesList(Term,Args,N1,R).

occurrences(Sub,Term,N,R) :-
    compound(Term), !,
    Term=..[_|Args],
    occurrencesList(Sub,Args,N,R).

occurrences(Term,Term,N,N1) :- N1 is N+1.
occurrences(_,_,N,N).

occurrencesList(_,[],N,N).
occurrencesList(Sub,[Arg|Args],N,N2) :-
    occurrencesList(Sub,Args,N,N1),
    occurrences(Sub,Arg,N1,N2).
```

(ii) Define a predicate *position*(*Subterm*,*Term*,*Position*), where *Position* is a list of argument positions identifying *Subterm* within *Term*. For example, the position of *X* in *2*sin(X)* is [2,1].

Ans.

```
position(Sub,Sub,[]).
position(Sub,Term,[N|T]) :-
    compound(Term), arg(N,Term,Arg),
    subterm(Sub,Arg), position(Sub,Arg,T).
```

(iv) Define *functor* and *arg* in terms of *univ*. How can the programs be used?

Ans.

```
%iv. functor and arg using UNIV
functr(Term,F,N) :-
    Term=..[F|T], length(T,N).

argUV(N,Term,Arg) :-
    Term=..[_|T], nth1(N,T,Arg).
```

```
?- ['hw03q02.pl'].
true.

?- subterm(a,f(X,Y)).
X = a ;
Y = a ;
false.

?- functor(length([q,w,e,r],8),F,N).
F = length,
N = 2.

?- arg(2,length([a,c,e,r],7),Y).
Y = 7.

?- occurrences(a,f(X,Y),Occs).
X = Y, Y = a,
Occs = 2 ;
Y = a,
Occs = 1 ;
X = a,
Occs = 1 ;
Occs = 0 ;
false.
```

```
?- ['hw03q02.pl'].
true.

?- functor(2*sin(x)+length([b,c,d],a),F,N).
F = (+),
N = 2.

?- position(a,2*sin(x)+length([b,c,d],a),L).
L = [2, 2] ;
false.

?- position(a,2*sin(a),L).
L = [2, 1] ;
false.

?- 
```

```
?- ['hw03q02.pl'].
true.

?- functor(length([a,c,e,r],7),F,N).
F = length,
N = 2.

?- functr(length([a,c,e,r],7),F,N).
F = length,
N = 2.

?- arg(2,length(X,Y,Z),What).
Y = What.

?- argUV(2,length(X,Y,Z),What).
Y = What ;
false.

?- 
```

(v) Rewrite Program 9.3 for *substitute* so that it uses *univ*.

```
%substitute(Old,New,OldTerm,NewTerm):  
%      NewTerm is the result of replacing all occurrences of Old in OldTerm by New.  
  
substitute(Old,New,Old,New).  
  
substitute(Old,_,Term,Term) :-  
    constant(Term), Term \= Old.  
  
substitute(Old,New,Term,Term1) :-  
    compound(Term), functor(Term,F,N),  
    functor(Term1,F,N),  
    substitute(N,Old,New,Term,Term1).  
  
substitute(N,Old,New,Term,Term1) :-  
    N > 0, arg(N,Term,Arg),  
    substitute(Old,New,Arg,Arg1),  
    arg(N,Term1,Arg1), N1 is N-1,  
    substitute(N1,Old,New,Term,Term1).  
substitute(0,Old,New,Term,Term1).  
  
constant(X) :- atomic(X).
```

Program 9.3 A program for substituting in a term.

Ans.

The screenshot shows a Prolog IDE with two windows. The left window, titled 'rahul: Assignment 03', contains the Prolog code for the `substitute` predicate and its test cases. The right window shows the results of the queries.

```
~/Desktop/Semester 02/6374.001 Computational Logic/Assignments/A...  rahul: Assignment 03  
hw03 x hw03 x a02q x a02q x hw03 x hw03 x hw03 x hw03 x hw03 x hw03 x temp x  
98 %-----  
99 % sub2(Old,New,Term0,TermN): other version of substitute,  
100 % Substituting Old from Term0, with New to give TermN.  
101  
102 sub2(Old,New,Old,New).  
103 sub2(Old,_,Term,Term) :-  
104     constant(Term), Term \== Old.  
105  
106 sub2(Old,New,Term0,TermN) :-  
107     Term0 =.. [Old|T],  
108     TermN =.. [New|T].  
109  
110 sub2(Old,New,Term0,TermN) :-  
111     functor(Term0,F,N), functor(TermN,F,N),  
112     Term0 =.. Lo, TermN =.. Ln,  
113     nth1(I,Lo,Old), nth1(I,Ln,New),  
114     length(Lo,G), I1 is I-1, I2 is I+1,  
115     ( I \= G ->  
116         slice(Lo,1,I1,L), slice(Ln,1,I1,L),  
117         slice(Lo,I2,G,R), slice(Ln,I2,G,R);  
118         slice(Lo,1,I1,L), slice(Ln,1,I1,L)).  
119  
120 %----- OLD PREDICATES -----  
121 % slice(L1,I,K,L2): L2 is the list of the elements of L1  
122 % between index I and index K (both included).  
123 % (list,integer,integer,list) (?,+,+,?)  
124  
125 slice([X|_],1,1,[X]).  
126 slice([X|Xs],1,K,[X|Ys]) :- K > 1,  
127     K1 is K-1, slice(Xs,1,K1,Ys).  
128 slice([_|Xs],I,K,Ys) :- I > 1,  
129     I1 is I-1, K1 is K-1, slice(Xs,I1,K1,Ys).  
130  
?- ['hw03q02.pl'].  
true.  
  
?- sub2(cat,dog,owns(jane,cat),T).  
T = owns(jane, dog) ;  
false.  
  
?- sub2(owns,loves,owns(jane,cat),T).  
T = loves(jane, cat) ;  
false.  
  
?- sub2(owns,Y,owns(jane,cat),likes(jane,cat)).  
Y = likes ;  
false.  
  
?- sub2(X,roger,owns(jane,cat),owns(roger,cat)).  
X = jane ;  
false.  
  
?-
```

3. Exercises 11.3 (i) and (ii)

(i) Define the system predicate `\==` using `==` and the cut-fail combination.

Ans.

```
notEq(X,Y) :-  
    neg(X==Y).
```

```
neg(P) :- P, !, fail.  
neg(_).
```

```
?- ['hw03q03.pl'].  
true.
```

```
?- notEq(e1,e1).  
false.
```

```
?- notEq(e1,e2).  
true.
```

```
?-  
```

(ii) Define `nonvar` using `var` and the cut-fail combination.

Ans.

```
nonVar(Term) :- var(Term), !, fail.  
nonVar(Term) :-  
    neg(var(Term)).
```

```
?- ['hw03q03.pl'].  
true.
```

```
?- nonVar(M1).  
false.
```

```
?- nonVar(m1).  
true.
```

```
?-  
```

4. Belgian Snake Problem

```
hw03q05.pl x hw03q04.pl x
1 % CS6374: Computational Logic - Homework 03
2 % Rahul Nalawade [RSN170330]
3 % Date: 2018-02-14
4
5 %----- Q4 -----
6 % snake(Pattern,Row,Column): represents belgian snake
7 % in Pattern forming a rectangle.
8 snake(_,[_],[_]).
9
10 snake(P,R,[C]) :-
11     makeFull(P,R,C,_), writeList(C).
12
13 snake(P,R,[C1,C2|Cs]) :-
14     makeFull(P,R,C1,P1),
15     makeFull(P1,R,Temp,P2),
16     reverse(Temp,C2),
17     writeList(C1), writeList(C2),
18     snake(P2,R,Cs).
19
20 %----- Q5 -----
21 % rotate([a,b,c],[b,c,a]) is true.
22 rotate([_],[_]).
23 rotate([X|T],R) :-
24     append(T,[X],R).
25
26 % makeFull(Pattern,Row,R,Pn): Fills Row with cyclic
27 % patterns to give R, with Pn as the last rotation.
28 makeFull(Plast,[_],[_],Plast).
29 makeFull([P|Ps],[_]|Rs,[P|Ls],Plast) :-
30     rotate([P|Ps],Pnext),
31     makeFull(Pnext,Rs,Ls,Plast).
32
33 % writes the List on a new-line
34 writeList([_]) :- nl.
35 writeList([H|T]) :- write(H), write(' '), writeList(T).
36
37 %----- OLD PREDICATES -----
38 % append(X,Y,Z): appends Y to X to give Z.
39 append([_],[_],[_]).
40 append(X,[_],X).
```

```
rahul: Assignment 03
rahul:Assignment 03$ swipl -q hw03q04.pl
?- snake([a,b,c,d,e],[_,_,_,_],[_,_,_]).
a b c d e
e d c b a
a b c d e
e d c b a
true ;
false.

?- snake([a,b,c,d,e],[_,_,_,_],[_,_,_]).
a b c d e
e d c b a
a b c d e
true ;
false.

?- snake([a,b,c,d,e],[_,_,_,_],[_,_,_]).
a b c d
c b a e
d e a b
true ;
false.

?- snake([a,b,c,d],[_,_,_,_],[_,_,_]).
a b c d a
b a d c b
c d a b c
true ;
false.

?- snake([a,b,c],[_,_,_,_],[_,_,_]).
a b c a b
a c b a c
b c a b c
true ;
false.

?- 
```

5. Program the N-Queen problem from the book.

```
hw03q05.pl x
1 % CS6374: Computational Logic - Homework 03
2 % Rahul Nalawade [RSN170330]
3 % Date: 2018-02-13
4
5 %----- Q5 -----
6 % queens(N,Queens): Queens is a placement that solves
7 % the N queens problem, represented as a permutation of
8 % the list of numbers [1,2,3,...,N].
9
10 queens(N,Qs) :-
11     range(1,N,Ns), permutation(Ns,Qs), safe(Qs).
12
13 % The placement Qs is safe/ valid.
14 safe([_]).
15 safe([Q|Qs]) :-
16     safe(Qs), \+ attack(Q,Qs).
17
18 attack(X,Xs) :- attack(X,1,Xs).
19 attack(X,N,[Y|_]) :-
20     X is Y+N; X is Y-N.
21 attack(X,N,[_]|Ys) :-
22     N1 is N+1, attack(X,N1,Ys).
23
24 % permutation(L1,P): P is a permutation of L1
25 permutation([_],[_]).
26 permutation(Xs,[Z|Zs]) :-
27     select(Z,Xs,Ys), permutation(Ys,Zs).
28
29 % range(M,N,List): List is list of integers [M,N], M<N.
30 range(N,N,[N]).
31 range(M,N,[M|Ns]) :-
32     M < N, M1 is M+1, range(M1,N,Ns).
33
34 % select(X,List,R): R is a List with one removed X
35 select(X,[X|T],T).
36 select(X,[H|T],[H|R]) :- X \= H,
37     select(X,T,R).
38
39 %----- Q6 -----
```

```
rahul: Assignment 03
rahul:Assignment 03$ swipl -q hw03q05.pl
true.

?- queens(5,S).
S = [1, 3, 5, 2, 4] ;
S = [1, 4, 2, 5, 3] ;
S = [2, 4, 1, 3, 5] ;
S = [2, 5, 3, 1, 4] ;
S = [3, 1, 4, 2, 5] ;
S = [3, 5, 2, 4, 1] ;
S = [4, 1, 3, 5, 2] ;
S = [4, 2, 5, 3, 1] ;
S = [5, 2, 4, 1, 3] ;
S = [5, 3, 1, 4, 2] ;
false.

?- queens(4,S).
S = [2, 4, 1, 3] ;
S = [3, 1, 4, 2] ;
false.

?- queens(4,[3,1,4,2]).
true ;
false.

?- queens(4,[3,1,2,4]).
false.

?- queens(3,S).
false.

?- queens(2,S).
false.

?- 
```

6. Write a Prolog program to solve cryptarithmic addition problems such as

```

  S E N D
+ M O R E
-----
M O N E Y

```

The solution is D = 7, E = 5, M = 1, N = 6, O = 0, R = 8, S = 9, Y = 2. Each letter should stand for a unique digit. If there is a solution, Prolog should return the list of letters and corresponding digits. If there is no solution, Prolog should report 'no'.

The screenshot shows a Prolog IDE with a dark theme. The main editor window displays a Prolog program for solving cryptarithms. The program includes a `search` predicate that generates possible digit assignments for the letters D, E, M, N, O, R, S, and Y, and checks if they satisfy the given addition problem. It also includes helper predicates like `matchV2V` and `select`. The right-hand pane shows the execution results for various queries, including the main `search` query which returns the solution: D=7, E=5, M=1, N=6, O=0, R=8, S=9, Y=2.

```

3 % Date: 2018-02-22
4
5 %----- Q6 -----
6 search :-
7     write('D E M N O R S Y'), nl,
8     Vars = [D,E,M,N,O,R,S,Y],
9     Values = [0,1,2,3,4,5,6,7,8,9],
10    matchV2V(Values, Vars),
11    S > 0, M > 0,
12    1000*S + 100*E + 10*N + D +
13    1000*M + 100*O + 10*R + E ==
14    10000*M + 1000*O + 100*N + 10*E + Y,
15    writeList(Vars).
16
17 % matchV2V(L1,L2): produces all possible combinations of variable
18 % of list L2, from list of values in L1.
19 matchV2V(_, []).
20 matchV2V(Values, [Vr|Vrs]):-
21     select(Vr, Values, NewValues),
22     matchV2V(NewValues, Vrs).
23
24 % select(X,List,R): R is a List with one removed X
25 select(X,[X|T],T).
26 select(X,[H|T],[H|R]) :- %X \= H,
27     select(X,T,R).
28
29 % writes the List on a new-line
30 writeList([]) :- nl.
31 writeList([H|T]) :- write(H), write(' '), writeList(T).
32
33 %----- := OPERATOR -----
34 %?- 2+3 == 6-1.
35 %true.
36
37 %?- 2+3 is 6-1.
38 %false.
39
40 % TIP: If you just need arithmetic comparison, use ==.
41 % If you want to capture the result of an evaluation, use is.
42

```

Execution results on the right:

```

?- ['hw03q06.pl'].
true.

?- matchV2V([1,2,3],[A,B]).
A = 1,
B = 2 ;
A = 1,
B = 3 ;
A = 2,
B = 1 ;
A = 2,
B = 3 ;
A = 3,
B = 1 ;
A = 3,
B = 2 ;
false.

?- select(A,[1,2,3],R).
A = 1,
R = [2, 3] ;
A = 2,
R = [1, 3] ;
A = 3,
R = [1, 2] ;
false.

?- search.
D E M N O R S Y
7 5 1 6 0 8 9 2
true ;
false.

?-

```

NOTE: Here, S and M are > 0. Hence, S,M can take values only from 1 to 9.

7. Solve the stable marriage problem in Exercise 14.1 (ii) pg. 261

Write a program to solve the stable marriage problem (Sedgewick, 1983), stated as follows:

Suppose there are N men and N women who want to get married. Each man has a list of all the women in his preferred order, and each woman has a list of all the men in her preferred order. The problem is to find a set of marriages that is stable.

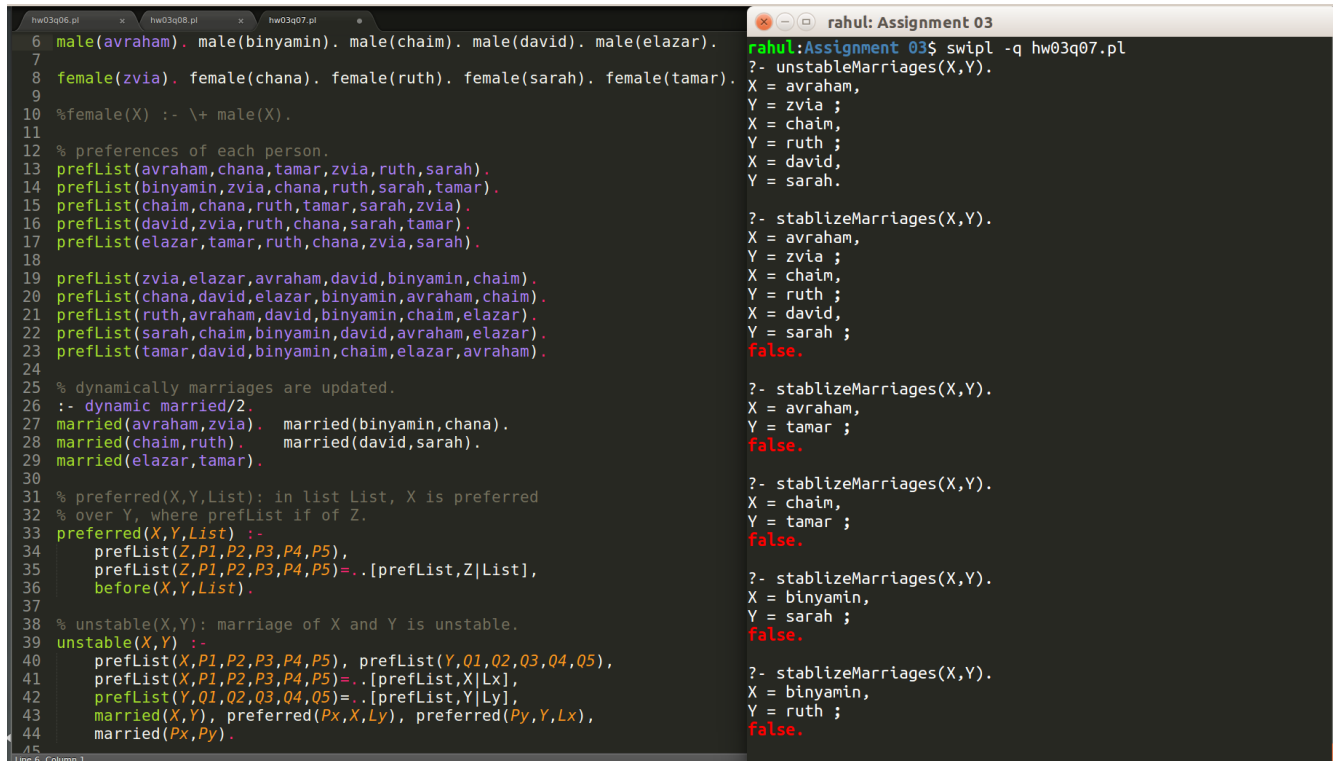
A pair of marriages is *unstable*, if there are a man and woman who prefer each other to their spouses. For example, consider the pair of marriages where David is married to Paula, and Jeremy is married to Judy. If David prefers Judy to Paula, and Judy prefers David to Jeremy, the pair of marriages is unstable. This pair would also be unstable if Jeremy preferred Paula to Judy, and Paula preferred Jeremy to David.

A set of marriages is stable if there is no pair of *unstable* marriages.

Your program should have as input lists of preferences. and produce as output a stable set of marriages. It is a theorem from graph theory that this is always possible. Test the program on the following five men and five women with their associated preferences:

avraham: chana tamar zvia ruth sarah
binyamin: zvia chana ruth sarah tamar
chaim: chana ruth tamar sarah zvia
david: zvia ruth chana sarah tamar
elazar: tamar ruth chana zvia sarah

zvia: elazar avraham david binyamin chaim
chana: david elazar binyamin avraham chaim
ruth: avraham david binyamin chaim elazar
sarah: chaim binyamin david avraham elazar
tamar: david binyamin chaim elazar avraham



```
hw03q06.pl x hw03q08.pl x hw03q07.pl
6 male(avraham). male(binyamin). male(chaim). male(david). male(elazar).
7
8 female(zvia). female(chana). female(ruth). female(sarah). female(tamar).
9
10 %female(X) :- \+ male(X).
11
12 % preferences of each person.
13 prefList(avraham, chana, tamar, zvia, ruth, sarah).
14 prefList(binyamin, zvia, chana, ruth, sarah, tamar).
15 prefList(chaim, chana, ruth, tamar, sarah, zvia).
16 prefList(david, zvia, ruth, chana, sarah, tamar).
17 prefList(elazar, tamar, ruth, chana, zvia, sarah).
18
19 prefList(zvia, elazar, avraham, david, binyamin, chaim).
20 prefList(chana, david, elazar, binyamin, avraham, chaim).
21 prefList(ruth, avraham, david, binyamin, chaim, elazar).
22 prefList(sarah, chaim, binyamin, david, avraham, elazar).
23 prefList(tamar, david, binyamin, chaim, elazar, avraham).
24
25 % dynamically marriages are updated.
26 :- dynamic married/2.
27 married(avraham, zvia). married(binyamin, chana).
28 married(chaim, ruth). married(david, sarah).
29 married(elazar, tamar).
30
31 % preferred(X,Y,List): in list List, X is preferred
32 % over Y, where prefList if of Z.
33 preferred(X,Y,List) :-
34     prefList(Z,P1,P2,P3,P4,P5),
35     prefList(Z,P1,P2,P3,P4,P5)=.[prefList,Z|List],
36     before(X,Y,List).
37
38 % unstable(X,Y): marriage of X and Y is unstable.
39 unstable(X,Y) :-
40     prefList(X,P1,P2,P3,P4,P5), prefList(Y,Q1,Q2,Q3,Q4,Q5),
41     prefList(X,P1,P2,P3,P4,P5)=.[prefList,X|Lx],
42     prefList(Y,Q1,Q2,Q3,Q4,Q5)=.[prefList,Y|Ly],
43     married(X,Y), preferred(Px,X,Ly), preferred(Py,Y,Lx),
44     married(Px,Py).
45
Line 6, Column 1

rahul: Assignment 03
rahul:Assignment 03$ swipl -q hw03q07.pl
?- unstableMarriages(X,Y).
X = avraham,
Y = zvia ;
X = chaim,
Y = ruth ;
X = david,
Y = sarah.

?- stabilizeMarriages(X,Y).
X = avraham,
Y = zvia ;
X = chaim,
Y = ruth ;
X = david,
Y = sarah ;
false.

?- stabilizeMarriages(X,Y).
X = avraham,
Y = tamar ;
false.

?- stabilizeMarriages(X,Y).
X = chaim,
Y = tamar ;
false.

?- stabilizeMarriages(X,Y).
X = binyamin,
Y = sarah ;
false.

?- stabilizeMarriages(X,Y).
X = binyamin,
Y = ruth ;
false.
```

Contd...

```

44 married(X,Y), preferred(Px,X,Py), preferred(Py,Y,Lx),
    married(Px,Py).
45
46 % unstableMarriages(X,Y): set of all unstable marriages.
47 unstableMarriages(X,Y) :-
48     setof((X,Y), unstable(X,Y), Pairs),
49     member((X,Y), Pairs).
50
51 % stabilize(X,Y): stabilize unstable marriage of X and Y.
52 stabilize(X,Y) :-
53     prefList(X,P1,P2,P3,P4,P5),
54     prefList(X,P1,P2,P3,P4,P5)=.[prefList,X|Lx],
55     prefList(Y,Q1,Q2,Q3,Q4,Q5),
56     prefList(Y,Q1,Q2,Q3,Q4,Q5)=.[prefList,Y|Ly],
57     married(X,Y), preferred(Px,X,Py), preferred(Py,Y,Lx),
58     married(Px,Py),
59     retract(married(X,Y)), retract(married(Px,Py)),
60     assert(married(Px,Y)), assert(married(X,Py)).
61
62 % stabilizeMarriages(X,Y): stabilize all unstable marriages.
63 stabilizeMarriages(X,Y) :-
64     unstableMarriages(X,Y),
65     stabilize(X,Y).
66
67 % TIP: you may need to call stabilizeMarriages(X,Y) multiple
68 % times, until it fails to give a solution.
69
70 %----- OLD PREDICATES -----
71 % before(X,Y,List): X is before Y in List
72 before(X,Y,List) :- justBefore(X,Y,List).
73 before(X,Y,List) :-
74     justBefore(Z,Y,List), before(X,Z,List).
75
76 justBefore(X,Y,[X,Y|_]).
77 justBefore(X,Y,[_|T]) :- justBefore(X,Y,T).
78
79 % writes the List on a new-line
80 writeList([]) :- nl.
81 writeList([H|T]) :- write(H), write(' '), writeList(T).
82 %-----

```

```

?- stabilizeMarriages(X,Y).
X = binyamin,
Y = sarah ;
false.

?- stabilizeMarriages(X,Y).
X = binyamin,
Y = ruth ;
false.

?- stabilizeMarriages(X,Y).
X = binyamin,
Y = chana ;
false.

?- stabilizeMarriages(X,Y).
X = elazar,
Y = chana ;
false.

?- stabilizeMarriages(X,Y).
false.

?- married(X,Y).
X = chain,
Y = sarah ;
X = avraham,
Y = ruth ;
X = binyamin,
Y = zvia ;
X = david,
Y = chana ;
X = elazar,
Y = tamar.

```

8. Program the block worlds problem described in the book (program the intelligent behavior using choose action on page 269). Assume that there are 3 locations p, q and r, and five blocks a, b, c, d, e. Generate a plan to go from initial configuration shown below to final configuration shown below.

Initial configuration:

```

b   e
a c d
-----
p q r

```

Final configuration:

```

e   a
d b c
-----
p q r

```

Ans.

Though, I've already produced a solution (as shown below), it seems not to be optimised. As I can find only 9 transformations (instead of 113) manually, I think the following algorithm may yield considerably less transformations:

Algorithm: IC, FC and CC: Initial, Final and Current Configuration.

1. find **freeBlocks(CC=IC)**, {here, [b,c,e]}
2. Make sure to place blocks closest to the table first, i.e. try placing [d,b,c] first.
 - How? And which ones first?
 - a. Find **baseBlocks(FC)**, which are not placed at proper base. {here, [d,b,c]}
 - b. match base blocks which are free. [d,b,c]
 - c. A = **findBase(b,FC)**, finds the base/stack of a block from FC;
 B = **findbase(b,CC)**. If A=B then d1; else d2. {here, A=q, B=p}
 - d1. Empty the stack 'q' to non 'p' stack; **emptyStack(q,r)**.
 - d2. Move 'b' to any non 'p' stack (say 'q'), and now **emptyStack(p,r)**
 - e. Place free block 'b' to 'q'.

Repeat 1 to 2e. Until all baseBlocks are placed.
Repeat this process for every next bottom-most layer.

Here, we can enumerate the moves following the above algorithm -

1. Bottom most level:

Placing b: toBlock(c,q,e), toPlace(b,p,q)	<pre> c e a b d p q r </pre>
Placing c: toBlock(c,e,a), toBlock(e,d,b), toBlock(d,r,e), toPlace(c,a,r)	<pre> d e a b c p q r </pre>
Placing d: toBlock(a,p,c), toPlace(d,e,p)	<pre> e a d b c p q r </pre>

2. Next level:

Placing e: toBlock(e,b,d)	<pre> e a d b c p q r </pre>
---------------------------	------------------------------

We might need to make it more specific, but placing level by level is the main idea here.

Anyways, here's the output. Might have some loops.

```

1 % CS6374: Computational Logic - Homework 03
2 % Rahul Malawade [RSN170330]
3 % Date: 2018-02-18
4
5 %----- Q8 -----
6 %State is represented by a list of relations on(X,Y),
7 % where X is a block and Y is a block or a place.
8
9 %?- testPlan(test2,Plan), length(Plan,L), writeList(Plan).
10 % transform([on(b,a),on(a,p),on(c,q),on(e,d),on(d,r)],[on(
11 %-----
12 testPlan(Name,Plan) :-
13     initial(Name,I), final(Name,F),
14     transform(I,F,Plan).
15
16 initial(test1,[on(a,b),on(b,p),on(c,r)]).
17 initial(test2,[on(b,a),on(a,p),on(c,q),on(e,d),on(d,r)]).
18
19 final(test1,[on(a,b),on(b,c),on(c,r)]).
20 final(test2,[on(e,d),on(d,p),on(b,q),on(a,c),on(c,r)]).
21
22 block(a). block(b). block(c). block(d). block(e).
23
24 place(p). place(q). place(r).
25
26 %-----
27 % transform(S1,S2,P): P is plan of actions that is produced
28 % to transform from state S1 to state S2.
29 transform(State1,State2,Plan) :-
30     transform(State1,State2,[State1],Plan).
31
32 %transform(State,State,Visited,[]).
33 transform(State1,State2,Visited,[]) :-
34     permute(State1,State2).
35
36 transform(State1,State2,Visited,[Action|Actions]) :-
37     %legalAction(Action,State1),
38     chooseAction(Action,State1,State2),
39     update(Action,State1,State),

```

```

?- ['hw03q08.pl'].
true.

?- testPlan(test2,Plan), length(Plan,L), writeList(Plan).
toBlock(b,a,c) toBlock(b,c,e) toBlock(a,p,c) toPlace(b,e,p) toBlock(b,p,a)
toPlace(e,d,p) toBlock(b,a,d) toBlock(b,d,e) toBlock(a,c,b) toBlock(a,b,d)
toBlock(b,e,a) toBlock(b,a,c) toBlock(a,d,b) toBlock(e,p,d) toBlock(a,b,e)
toPlace(b,c,p) toBlock(b,p,a) toPlace(c,q,p) toPlace(b,a,q) toBlock(a,e,c)
toBlock(b,q,a) toPlace(e,d,q) toBlock(b,a,d) toBlock(b,d,e) toBlock(a,c,b)
toBlock(a,b,d) toBlock(b,e,a) toBlock(b,a,c) toBlock(a,d,b) toBlock(e,q,d)
toPlace(a,b,q) toBlock(b,c,a) toBlock(b,a,e) toBlock(c,p,a) toPlace(b,e,p)
toBlock(b,p,c) toPlace(e,d,p) toBlock(b,c,d) toBlock(b,d,e) toBlock(c,a,b)
toBlock(a,q,c) toPlace(d,r,q) toPlace(a,c,r) toBlock(a,r,d) toPlace(c,b,r)
toBlock(a,d,c) toBlock(b,e,a) toBlock(e,p,d) toPlace(b,a,p) toBlock(b,p,e)
toPlace(a,c,p) toBlock(b,e,a) toBlock(b,a,c) toBlock(a,p,b) toPlace(e,d,p)
toBlock(a,b,d) toBlock(b,c,a) toBlock(c,r,b) toPlace(e,p,r) toPlace(c,b,p)
toBlock(b,a,c) toBlock(b,c,e) toBlock(a,d,c) toBlock(b,e,a) toBlock(e,r,d)
toPlace(b,a,r) toBlock(b,r,e) toPlace(a,c,r) toBlock(b,e,a) toBlock(b,a,c)
toBlock(a,r,b) toPlace(e,d,r) toBlock(a,b,e) toBlock(b,c,a) toBlock(b,a,d)
toBlock(a,e,c) toBlock(a,c,b) toBlock(c,p,a) toPlace(e,r,p) toPlace(c,a,r)
toBlock(a,b,c) toBlock(a,c,e) toBlock(b,d,a) toBlock(b,a,c) toBlock(d,q,a)
toPlace(b,c,q) toBlock(b,q,d) toPlace(c,r,q) toPlace(b,d,r) toBlock(b,r,c)
toPlace(d,a,r) toBlock(b,c,a) toBlock(b,a,d) toBlock(a,e,b) toBlock(c,q,a)
toPlace(e,p,q) toPlace(c,a,p) toBlock(a,b,e) toBlock(b,d,a) toBlock(b,a,c)
toBlock(d,r,a) toPlace(b,c,r) toBlock(b,r,d) toPlace(c,p,r) toPlace(b,d,p)
toBlock(b,p,c) toPlace(d,a,p) toBlock(b,c,a) toBlock(b,a,d) toBlock(a,e,c)
toBlock(b,d,a) toBlock(e,q,d) toPlace(b,a,q)
Plan = [toBlock(b, a, c), toBlock(b, c, e), toBlock(a, p, c), toPlace(b, e,
p), toBlock(b, p, a), toPlace(e, d, p), toBlock(b, a, d), toBlock(b, d, e)
, toBlock(..., ..., ...)]...,
L = 113 .

?-

```

9. Consider the missionary-cannibal problem. Three missionaries and three cannibals come to a river that they want to cross and find a boat that holds two. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten.

Think of this problem as a planning problem and program it in Prolog. You should print a sequence of moves (one per line) that lists the people crossing the river at each step. You can represent the 3 cannibals as c1, c2 and c3, and the three missionaries as m1, m2 and m3. For example, the move:

c1 m1

means that cannibal c1 and missionary m1 went from the bank where the boat is to the other bank. Keep in mind that the boat is always needed to go from one side to another.

Ans.

Comma ',' in the output line as shown below is the river, partitioning left(before crossing) and right(after crossing) sides.

```

146
147 % Termination of Search: must occur when attempts to search a RI
148 searchMovesR2L([L1,R1],[L2,R2], Moves) :-
149     sort(L1,L), sort(L2,L), sort(R1,R), sort(R2,R),
150     length(Moves,M),
151     write('No of Moves = '), write(M), nl,
152     writeMoves(Moves,0).
153
154 %----- WRITING OUTPUT -----
155
156 % Writes Moves in Zig-Zag manner.
157 writeMoves([],_) :- nl.
158 writeMoves([[[L2s,R2s],Moved,[L1s,R1s]]|T],Flag) :-
159     (Flag == 0 ->
160         writeMoves(T,1),
161         writeList(L1s), write(' '), writeList(R1s),
162         write(' -> '), writeList(Moved), write(' -> '),
163         writeList(L2s), write(' '), writeList(R2s), nl;
164
165         writeMoves(T,0),
166         writeList(L2s), write(' '), writeList(R2s),
167         write(' <- '), writeList(Moved), write(' <- '),
168         writeList(L1s), write(' '), writeList(R1s), nl).
169
170 % writes the List [H|T].
171 writeList([]).
172 writeList([H|T]) :- write(H), write(' '), writeList(T).
173
174 %----- MAIN PREDICATE -----
175 % Represent a state as [L,R]
176 % start: [[c1,c2,c3,m1,m2,m3],[]]
177 % goal: [[],[c1,c2,c3,m1,m2,m3]]
178 % NOTE - Start and Goal States must be sorted.
179
180 search :-
181     Start = [[c1,c2,c3,m1,m2,m3],[]],
182     Goal = [[],[c1,c2,c3,m1,m2,m3]],
183     searchMovesL2R(Start,Goal,[Start],_).
184

```

```

?- ['hw03q09.pl'].
true.

?- search.
No of Moves = 11

c1 c2 c3 m1 m2 m3 , -> c1 c2 -> c3 m1 m2 m3 , c1 c2
c1 c3 m1 m2 m3 , c2 <- c1 xx <- c3 m1 m2 m3 , c1 c2
c1 c3 m1 m2 m3 , c2 -> c1 c3 -> m1 m2 m3 , c1 c2 c3
c1 m1 m2 m3 , c2 c3 <- c1 xx <- m1 m2 m3 , c1 c2 c3
c1 m1 m2 m3 , c2 c3 -> m1 m2 -> c1 m3 , c2 c3 m1 m2
c1 c2 m1 m3 , c3 m2 <- m1 c2 <- c1 m3 , c2 c3 m1 m2
c1 c2 m1 m3 , c3 m2 -> m1 m3 -> c1 c2 , c3 m1 m2 m3
c1 c2 c3 , m1 m2 m3 <- c3 xx <- c1 c2 , c3 m1 m2 m3
c1 c2 c3 , m1 m2 m3 -> c1 c2 -> c3 , c1 c2 m1 m2 m3
c3 m1 , c1 c2 m2 m3 <- m1 xx <- c3 , c1 c2 m1 m2 m3
c3 m1 , c1 c2 m2 m3 -> m1 c3 -> , c1 c2 c3 m1 m2 m3
true .

?-

```