```
write_on_file(File ,Text):-
      open(File, append, Stream) ,
      write(Stream,Text),
      nl(Stream),
      close(Stream).

r2(File):-
      open(File,read,Stream),
      get_char(Stream, Char1),
      process_the_stream(Char1,Stream),
      close(Stream).
process_the_stream(end_of_file , _ ) :- ! .
process_the_stream(Char,Stream):-
      write(Char),
      get_char(Stream,Char2),
      process_the_stream(Char2,Stream).

mymember(X,[X|_]).
mymember(X,[_|T]) :-
mymember(X,T) .
mem(X,[H|T]):-
X = H ;
mem(X,T).

list_len([],0) .
list_len([_|T],N) :-
list_len(T,M) ,
N is M+1 .


list_sum([],0) .
list_sum([H|T] , S ) :-
list_sum(T,S1),
S is S1 + H .

list_avg([H|T],V) :-
list_len([H|T],N),
list_sum([H|T],S),
V is S / N .

is_even(L) :-
list_len(L,X1),
0 =:= mod(X1,2).

is_odd(L) :-
list_len(L,X1),
1 =:= mod(X1,2).

even([]).
even([_,_|T]):-
even(T).

odd([_]).
odd([_,_|T]):-
odd(T).

check_list(L) :-
even(L),write("even");
odd(L) , write("odd").
```

```
list_con([],L,L) .
list_con( [H|T] ,L,[H|T1] ) :-
list_con(T,L,T1) .

union([],L,L).
union([H|T],L2,L3):-
mem(H,L2) , union(T,L2,L3).
union([H|T],L2,[H|T3]):-
\+ mem(H,L2) , union(T,L2,T3).

intersection([],_,[]).
intersection([H|T],L2,[H|T3]):-
mem(H,L2) , intersection(T,L2,T3).

intersection([H|T],L2,L3):-
\+ mem(H,L2) , intersection(T,L2,L3).

split([],[],[]).
split([A],[],[A]).
split([A,B|T],[A|T2],[B|T3]):-
split(T,T2,T3).

mergeSort([],[]).
mergeSort([A],[A]).
mergeSort(L,SL):-
split(L,A,B),
mergeSort(A,SA),
mergeSort(B,SB),
merge(SA,SB,SL).

merge(A,[],A).
merge([],B,B).
merge([A|T],[E|T2],[A|T3]):-
A =< E ,
merge(T,[E|T2],T3).
merge([A|T],[E|T2],[E|T3]):-
E < A ,
merge(T2,[A|T],T3).
```

```
max(N1,N2,N1) :- N1 >= N2 .
max(N1,N2,N2) :- N2 > N1 .

list_max([M],M) .
list_max([H|T],M):-
list_max(T,M1),
max(H,M1,M).

min(N1,N2,N1) :- N1 =< N2 .
min(N1,N2,N2) :- N2 < N1 .

list_min([M],M).
list_min([H|T],M) :-
list_min(T,M1),
min(H,M1,M).

list_list_divide([],[],[]).
list_list_divide([N],[N],[]).
list_list_divide([N1,N2|T],[N1|T1],[N2|T2]):-
list_list_divide(T,T1,T2).
list_list_divide([],[],[]).
list_list_divide([M],[M],[]).
list_list_divide([N1,N2|T],[N1|T1],[N2|T2]):-
    list_list_divide(T,T1,T2).



list_con([],L1,L2,T) :-
    list_con(L1,L2,T) .

list_con([H|T],L1,L2,[H|T1]):-
    list_con(T,L1,L2,T1).




list_divide(L,X,Y):-

    even(L),
        list_con(X,Y,L),
        list_len(X,N1),
        list_len(Y,N2),
        N1 = N2 ;
    odd(L),

        list_con(X,Y,L),
        list_len(X,N1),
        list_len(Y,N2),
        N1 =:= N2+1 .

list_divide(L,X,Y,Z):-
    list_con(X,Y,Z,L),
    list_len(X,N1),
    list_len(Y,N2),
    list_len(Z,N3),
    N1 = N2 ,
N2 = N3 ,
!.

print_list([]) .
print_list([H|T]):-
write(H),
write(' '),
print_list(T),
!.


delete_last_three_elements(L,L1) :-
list_len(L,N1),
N1 > 3 ,
list_con(L1,X,L),
list_len(X,N),
N = 3 .
delete_first_three_elements(L,L1) :-
list_len(L,N1),
N1 > 3 ,
list_con(X,L1,L),
list_len(X,N),
N = 3 ,
!.
delete_first_and_last_three_elements(L,L1) :-
list_len(L,N) ,
N >= 6 ,
delete_last_three_elements(L,X),
delete_first_three_elements(X,L1),
!.

last_item(I,L) :-
list_len(L,N1),
N1 > 1 ,
list_con(_,I,L),
list_len(I,N),
N = 1 ,
!.

delete_item(I,[I|T], T) .
delete_item(I,[H|T],[H|T1]):-
delete_item(I,T,T1).

insert(I,L,NL) :-
delete_item(I,NL,L).


swap([N1,N2|T],[N2,N1|T]) :- N1 > N2 .
swap([H|T],[H|T1]) :- swap(T,T1) .

bubble_sort(L,SL) :-
swap(L,TL),
print_list(TL),
! ,
bubble_sort(TL,SL).

bubble_sort(Sl,Sl).

print_list([]) :- nl .
print_list([H|T]) :-
```

```
write(H) ,
write(" "),
print_list(T).

listCon([],L,L).
listCon([H|T],L,[H|T1]):-
listCon(T,L,T1).
lastElement([H|T],X):-




%% move left in the top row

move([X1,0,X3, X4,X5,X6, X7,X8,X9],
[0,X1,X3, X4,X5,X6, X7,X8,X9]).
move([X1,X2,0, X4,X5,X6, X7,X8,X9],
[X1,0,X2, X4,X5,X6, X7,X8,X9]).

%% move left in the middle row
move([X1,X2,X3, X4,0,X6,X7,X8,X9],
[X1,X2,X3, 0,X4,X6,X7,X8,X9]).
move([X1,X2,X3, X4,X5,0,X7,X8,X9],
[X1,X2,X3, X4,0,X5,X7,X8,X9]).

%% move left in the bottom row
move([X1,X2,X3, X4,X5,X6, X7,0,X9],
[X1,X2,X3, X4,X5,X6, 0,X7,X9]).
move([X1,X2,X3, X4,X5,X6, X7,X8,0],
[X1,X2,X3, X4,X5,X6, X7,0,X8]).

%% move right in the top row
```

```
move([0,X2,X3, X4,X5,X6, X7,X8,X9],
[X2,0,X3, X4,X5,X6, X7,X8,X9]).
move([X1,0,X3, X4,X5,X6, X7,X8,X9],
[X1,X3,0, X4,X5,X6, X7,X8,X9]).

%% move right in the middle row
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
[X1,X2,X3, X5,0,X6, X7,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
[X1,X2,X3, X4,X6,0, X7,X8,X9]).

%% move right in the bottom row
move([X1,X2,X3, X4,X5,X6,0,X8,X9],
[X1,X2,X3, X4,X5,X6,X8,0,X9]).
move([X1,X2,X3, X4,X5,X6,X7,0,X9],
[X1,X2,X3, X4,X5,X6,X7,X9,0]).

%% move up from the middle row
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
[0,X2,X3, X1,X5,X6, X7,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
[X1,0,X3, X4,X2,X6, X7,X8,X9]).
move([X1,X2,X3, X4,X5,0, X7,X8,X9],
[X1,X2,0, X4,X5,X3, X7,X8,X9]).

%% move up from the bottom row
move([X1,X2,X3, X4,X5,X6, 0,X8,X9],
[X1,X2,X3, 0,X5,X6, X4,X8,X9]).
move([X1,X2,X3, X4,X5,X6, X7,0,X9],
[X1,X2,X3, X4,0,X6, X7,X5,X9]).
move([X1,X2,X3, X4,X5,X6, X7,X8,0],
[X1,X2,X3, X4,X5,0, X7,X8,X6]).
```

```
%% move down from the top row
move([0,X2,X3, X4,X5,X6, X7,X8,X9],
[X4,X2,X3, 0,X5,X6, X7,X8,X9]).
move([X1,0,X3, X4,X5,X6, X7,X8,X9],
[X1,X5,X3, X4,0,X6, X7,X8,X9]).
move([X1,X2,0, X4,X5,X6, X7,X8,X9],
[X1,X2,X6, X4,X5,0, X7,X8,X9]).

%% move down from the middle row
move([X1,X2,X3, 0,X5,X6, X7,X8,X9],
[X1,X2,X3, X7,X5,X6, 0,X8,X9]).
move([X1,X2,X3, X4,0,X6, X7,X8,X9],
[X1,X2,X3, X4,X8,X6, X7,0,X9]).
move([X1,X2,X3, X4,X5,0, X7,X8,X9],
[X1,X2,X3, X4,X5,X9, X7,X8,0]).


dfs(State,State,PATH,PATH) :- showSolutionPath(PATH).
dfs(State,Goal,CHECKED,PATH):-
move(State,State2),
\+member(State2, CHECKED),
dfs(State2,Goal,[State2|CHECKED],PATH).

dfs(State,Goal,Path):-
dfs(State,Goal,[],Path).

showPuzzelState([X1,X2,X3, X4,X5,X6, X7,X8,X9]):-
write("-------"),nl,
write('|'),write(X1),write('|'),write(X2),write('|'),write(X3
),write('|'),nl,
write("-------"),nl,
write('|'),write(X4),write('|'),write(X5),write('|'),write(X6
),write('|'),nl,
write("-------"),nl,
write('|'),write(X7),write('|'),write(X8),write('|'),write(X9
),write('|'),nl,
write("-------"),nl.

showSolutionPath([]):- write("Done"),nl.
showSolutionPath([H|T]):-
showPuzzelState(H),
showSolutionPath(T).
```

```
edge(1,[2,3]).

edge(2,[4,7]).
edge(3,[5,6]).
edge(4,[8,9]).
edge(7,[]).
edge(8,[]).
edge(9,[]).
edge(5,[]).
edge(6,[]).

breadthfirst([], List, List).
breadthfirst([H|L1], List, [H|L3]) :-
breadthfirst(L1, List, L3).

sol(X, [F|T]) :-
edge(F, Y),
breadthfirst(T, Y, Z),
write(F), nl,
sol(X, Z).
% goal
% sol(9, [1]).


graph([0,0,0,0], [1,0,1,0]) :- write('[1,0,1,0]\n').%

graph([0,0,0,1], [1,1,0,1]) :- write('[1,1,0,1]\n').%
graph([0,0,1,0], [1,0,1,1]) :- write('[1,0,1,1]\n').%
graph([0,0,1,0], [1,1,1,0]) :- write('[1,1,1,0]\n').%
graph([0,1,0,0], [1,1,0,1]) :- write('[1,1,0,1]\n').%
graph([0,1,0,1], [1,1,1,1]) :- write('[1,1,1,1]\n').
graph([1,0,1,0], [0,0,1,0]) :- write('[0,0,1,0]\n').%
graph([1,0,1,1], [0,0,0,1]) :- write('[0,0,0,1]\n').%
graph([1,1,0,1], [0,1,0,1]) :- write('[0,1,0,1]\n').
graph([1,1,1,0], [0,1,0,0]) :- write('[0,1,0,0]\n').%


%****** DFS ******
dfs_2(D, D, P, P).

% S=Source, D=Destination, C=Checked, P=Path
dfs_2(S, D, C, P) :-
    graph(S, S2),
    \+ member(S2, C),
    dfs_2(S2, D, [S2 | C], P).
```

```
% S=Source, D=Destination, P_r=Path_reversed,
P_o=Path_ordered
dfs(S, D, P) :-
    dfs_2(S, D, [], P_r),
    reverse(P_r, P_o),
    P=[S | P_o].
%****** DFS ******


inform(D, Loc1, Loc2) :-

    nl,
    write('Move disk\t'), write(D),
    write('\tfrom\t'), write(Loc1),
    write('\tto\t'), write(Loc2).


move([H], A, C, _):-
    inform(H, A, C), !.

move([H|T], A, C, B):-
    move(T, A, B, C),
    inform(H, A, C),
    move(T, B, C, A).


hanoi(L) :-
    write('HANOI TOWERS PROBLEM:\n'),
    move(L, left, right, middle).
%****** /Hanoi ******%
```