** ACE**

**Engineering College**

**An AUTONOMOUS Institution**

Ghatkesar, Medchal (Dist), Hyderabad, Telangana State – 501 301

(NBA Accredited B.Tech Courses Accredited NAAC with A Grade 3.20 CGPA

**DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

# CM409PC: PROLOG/ LISP/ PYSWIP

**B.Tech. II Year II Sem. L T P C**

# 0 0 2 1

**List of Programs:**

1. Write simple fact for following:
   1. Ram likes mango.
   2. Seema is a girl.
   3. Bill likes Cindy.
   4. Rose is red.
   5. John owns gold
2. Write predicates one converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.
3. Write a program to solve the Monkey Banana problem
4. WAP in turbo prolog for medical diagnosis and show the advantages and disadvantages of green and red cuts.
5. Write a program to solve the 4-Queen problem.
6. Write a program to solve traveling salesman problems.
7. Write a program to solve water jug problems using Prolog.
8. Write simple Prolog functions such as the following. Take into account lists which are too short.

-- remove the Nth item from the list. -- insert as the Nth item.

1. Assume the prolog predicate gt(A, B) is true when A is greater than B. Use this predicate to define the predicate addLeaf(Tree, X, NewTree) which is true if NewTree is the Tree produced by adding the item X in a leaf node. Tree and NewTree are binary search trees. The empty tree is represented by the atom nil.
2. Write a Prolog predicate, countLists(Alist, Ne, Nl), using accumulators, that is true when Nl is the number of items that are listed at the top level of Alist and Ne is the number of empty lists. Suggestion: First try to count the lists, or empty lists, then modify by adding the other counter.
3. Define a predicate memCount(AList,Blist,Count) that is true if Alist occurs Count times within Blist. Define without using an accumulator. Use "not" as defined in utilities.pro, to make similar cases are unique, or else you may get more than one count as an answer.

Examples:

memCount(a,[b,a],N). N = 1 ;

no memCount(a,[b,[a,a,[a],c],a],N). N = 4 ;

no memCount([a],[b,[a,a,[a],c],a],N). N = 1 ;

No

# REFERENCE BOOK:

1. PROLOG: Programming for Artificial Intelligence, 3e, by BRATKO, WILEY

1. Write simple fact for following:
   1. Ram likes mango.
   2. Seema is a girl.
   3. Bill likes Cindy.
   4. Rose is red.
   5. John owns gold

**INPUT:**

% Facts

1. Ram likes mango.

2. Seema is a girl.

3. Bill likes Cindy.

4. Rose is red.

5. John owns gold.

% Clauses

likes(ram ,mango).

girl(seema).

red(rose).

likes(bill ,cindy).

owns(john ,gold).

**OUTPUT:**

% Queries

?-likes(ram,What).

What= mango

?-likes(Who,cindy).

Who= cindy

?-red(What).

What= rose

?-owns(Who,What).

Who= john

What= gold

1. Write predicates one converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.

**SOLUTION:**

% Production rules:

c\_to\_f

f is c \* 9 / 5 + 32

freezing f < = 32

% Rules:

c\_to\_f(C,F) :-

F is C \* 9 / 5 + 32.

freezing(F) :-

F =< 32.

**OUTPUT:**

% Queries :

?- c\_to\_f(100,X).

X = 212

Yes

?- freezing(15).

Yes

?- freezing(45).

No

1. Write a program to solve the Monkey Banana problem

/\* Description:

Imagine a room containing a monkey, chair and some bananas. That have been hanged from the center of ceiling. If the monkey is clever enough he can reach the bananas by placing the chair directly below the bananas and climb on the chair .

The problem is to prove the monkey can reach the bananas.

The monkey can perform the following actions:

1) Walk on the floor

2) Climb the box

3) Push the box around(if it is beside the box).

4) Grasp the banana if it is standing on the box directly under the banana.

\*/

% Production rules:

can\_reach 🡪 clever,close.

get\_on: 🡪 can\_climb.

under 🡪 in room,in\_room, in\_room,can\_climb.

Close 🡪 get\_on,under | tall

% Clauses:

in\_room(bananas).

in\_room(chair).

in\_room(monkey).

clever(monkey).

can\_climb(monkey, chair).

tall(chair).

can\_move(monkey, chair, bananas).

can\_reach(X, Y):-clever(X),close(X, Y).

get\_on(X,Y):-

can\_climb(X,Y).

under(Y,Z):-

in\_room(X),in\_room(Y),

in\_room(Z),can\_climb(X,Y,Z).

close(X,Z):-

get\_on(X,Y), under(Y,Z);

tall(Y).

OUTPUT:

% Queries:

?- can\_reach(A, B).

A = monkey.

B = banana.

?- can\_reach(monkey, banana).

Yes.

1. WAP in turbo prolog for medical diagnosis and show the advantages and disadvantages of green and red cuts.

SOLUTION:

1. /\* Description:
2. This object of this famous puzzle is to move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk. The following diagram depicts the starting setup for N=3 disks.
3. \*/
4. % Production rules:
5. hanoi(N) 🡪 move(N,left,middle,right).
6. move(1,A,\_,C) 🡪 inform(A,C),fail.
7. move(N,A,B,C) 🡪 N1=N-1,move(N1,A,C,B),inform(A,C),move(N1,B,A,C).
8. % Domains:
9. loc =right;middle;left
10. % Predicates:
11. hanoi(integer)
12. move(integer,loc,loc,loc)
13. inform(loc,loc)
14. % Clauses:
15. hanoi(N):-
16. move(N,left,middle,right).
17. move(1,A,\_,C):-
18. inform(A,C),!.
19. move(N,A,B,C):-
20. N1=N-1,
21. move(N1,A,C,B),
22. inform(A,C),
23. move(N1,B,A,C).
24. inform(Loc1, Loc2):-
25. write("\nMove a disk from ", Loc1, " to ", Loc2).

OUTPUT:

% Queries:

?- can\_reach(A, B).

A = monkey.

B = banana.

?- can\_reach(monkey, banana).

Yes.

1. Write a program to solve the 4-Queen problem.

SOLUTION:

/\* Description:

In the 4 Queens problem the object is to place 4 queens on a chessboard in such a way that no queens can capture a piece. This means that no two queens may be placed on the same row, column, or diagonal.

\*/

% Domains:

queen = q(integer, integer)

queens = queen\*

freelist = integer\*

board = board(queens, freelist, freelist, freelist, freelist)

% Predicates:

nondeterm placeN(integer, board, board)

nondeterm place\_a\_queen(integer, board, board)

nondeterm nqueens(integer)

nondeterm makelist(integer, freelist)

nondeterm findandremove(integer, freelist, freelist)

nextrow(integer, freelist, freelist)

% Clauses

nqueens(N):-

makelist(N,L),

Diagonal=N\*2-1,

makelist(Diagonal,LL),

placeN(N,board([],L,L,LL,LL),Final),

write(Final).

placeN(\_,board(D,[],[],D1,D2),board(D,[],[],D1,D2)):-!.

placeN(N,Board1,Result):-

place\_a\_queen(N,Board1,Board2),

placeN(N,Board2,Result).

place\_a\_queen(N,

board(Queens,Rows,Columns,Diag1,Diag2),

board([q(R,C)|Queens],NewR,NewC,NewD1,NewD2)):-

nextrow(R,Rows,NewR),

findandremove(C,Columns,NewC),

D1=N+C-R,findandremove(D1,Diag1,NewD1),

D2=R+C-1,findandremove(D2,Diag2,NewD2).

findandremove(X,[X|Rest],Rest).

findandremove(X,[Y|Rest],[Y|Tail]):-

findandremove(X,Rest,Tail).

makelist(1,[1]).

makelist(N,[N|Rest]) :-

N1=N-1,makelist(N1,Rest).

nextrow(Row,[Row|Rest],Rest).

OUTPUT:

% Goal

nqueens(4),nl.

board([q(1,2),q(2,4),q(3,1),q(4,3),[],[],[7,4,1],[7,4,1])

yes

1. Write a program to solve traveling salesman problems.

SOLUTION:

/\* Description:

For example, there are four cities(Kansas City,Houston,Gordon and Tampa).

-> The distance between Kansas City and Houston is 120.

-> The distance between Kansas City and Tampa is 80.

-> The distance between Houston and Gordon is 100.

\*/

% Production Rules:-

route(Town1,Town2,Distance)🡪 road(Town1,Town2,Distance).

route(Town1,Town2,Distance)🡪 road(Town1,X,Dist1),

route(X,Town2,Dist2),

Distance=Dist1+Dist2,

% Domains

town = symbol

distance = integer

% Predicates

nondeterm road(town,town,distance)

nondeterm route(town,town,distance)

% Clauses

road("tampa","houston",200).

road("gordon","tampa",300).

road("houston","gordon",100).

road("houston","kansas\_city",120).

road("gordon","kansas\_city",130).

route(Town1,Town2,Distance):-

road(Town1,Town2,Distance).

route(Town1,Town2,Distance):-

road(Town1,X,Dist1),

route(X,Town2,Dist2),

Distance=Dist1+Dist2,

!.

OUTPUT:

% Goal

route("tampa", "kansas\_city", X),

write("Distance from Tampa to Kansas City is ",X),nl.

Distance from Tampa to Kansas City is 320

X=320

1 Solution

1. Write a program to solve water jug problems using Prolog.

SOLUTION:

/\* Description:

"You are given two jugs, a 4-gallon one and a 3-gallon one. Neither have any measuring markers on it. There is a tap that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug?".

\*/

/\* Production Rules:-

R1: (x,y) --> (4,y) if x < 4

R2: (x,y) --> (x,3) if y < 3

R3: (x,y) --> (x-d,y) if x > 0

R4: (x,y) --> (x,y-d) if y > 0

R5: (x,y) --> (0,y) if x > 0

R6: (x,y) --> (x,0) if y > 0

R7: (x,y) --> (4,y-(4-x)) if x+y >= 4 and y > 0

R8: (x,y) --> (x-(3-y),y) if x+y >= 3 and x > 0

R9: (x,y) --> (x+y,0) if x+y =< 4 and y > 0

R10: (x,y) --> (0,x+y) if x+y =< 3 and x > 0

\*/

%database

visited\_state(integer,integer).

%predicates

state(integer,integer).

%clauses

state(2,0).

state(X,Y):- X < 4,

not(visited\_state(4,Y)),

assert(visited\_state(X,Y)),

write("Fill the 4-Gallon Jug: (",X,",",Y,") --> (", 4,",",Y,")\n"),

state(4,Y).

state(X,Y):- Y < 3,

not(visited\_state(X,3)),

assert(visited\_state(X,Y)),

write("Fill the 3-Gallon Jug: (", X,",",Y,") --> (", X,",",3,")\n"),

state(X,3).

state(X,Y):- X > 0,

not(visited\_state(0,Y)),

assert(visited\_state(X,Y)),

write("Empty the 4-Gallon jug on ground: (", X,",",Y,") --> (", 0,",",Y,")\n"),

state(0,Y).

state(X,Y):- Y > 0,

not(visited\_state(X,0)),

assert(visited\_state(X,0)),

write("Empty the 3-Gallon jug on ground: (", X,",",Y,") --> (", X,",",0,")\n"),

state(X,0).

state(X,Y):- X + Y >= 4,

Y > 0,

NEW\_Y = Y - (4 - X),

not(visited\_state(4,NEW\_Y)),

assert(visited\_state(X,Y)),

write("Pour water from 3-Gallon jug to 4-gallon until it is full: (", X,",",Y,") --> (", 4,",",NEW\_Y,")\n"),

state(4,NEW\_Y).

state(X,Y):- X + Y >=3,

X > 0,

NEW\_X = X - (3 - Y),

not(visited\_state(X,3)),

assert(visited\_state(X,Y)),

write("Pour water from 4-Gallon jug to 3-gallon until it is full: (", X,",",Y,") --> (", NEW\_X,",",3,")\n"),

state(NEW\_X,3).

state(X,Y):- X + Y>=4,

Y > 0,

NEW\_X = X + Y,

not(visited\_state(NEW\_X,0)),

assert(visited\_state(X,Y)),

write("Pour all the water from 3-Gallon jug to 4-gallon: (", X,",",Y,") --> (", NEW\_X,",",0,")\n"),

state(NEW\_X,0).

state(X,Y):- X+Y >=3,

X > 0,

NEW\_Y = X + Y,

not(visited\_state(0,NEW\_Y)),

assert(visited\_state(X,Y)),

write("Pour all the water from 4-Gallon jug to 3-gallon: (", X,",",Y,") --> (", 0,",",NEW\_Y,")\n"),

state(0,NEW\_Y).

state(0,2):- not(visited\_state(2,0)),

assert(visited\_state(0,2)),

write("Pour 2 gallons from 3-Gallon jug to 4-gallon: (", 0,",",2,") --> (", 2,",",0,")\n"),

state(2,0).

state(2,Y):- not(visited\_state(0,Y)),

assert(visited\_state(2,Y)),

write("Empty 2 gallons from 4-Gallon jug on the ground: (", 2,",",Y,") --> (", 0,",",Y,")\n"),

state(0,Y).

goal:-

makewindow(1,2,3,"4-3 Water Jug Problem",0,0,25,80),

state(0,0).

OUTPUT:

% Goal:-

makewindow(1,2,3,"4-3 Water Jug Problem",0,0,25,80),

state(0,0).

+-----------------------------4-3 Water Jug Problem--------------------------+

| Fill the 4-Gallon Jug: (0,0) --> (4,0) |

| Fill the 3-Gallon Jug: (4,0) --> (4,3) |

| Empty the 4-Gallon jug on ground: (4,3) --> (0,3) |

| Pour all the water from 3-Gallon jug to 4-gallon: (0,3) --> (3,0) |

| Fill the 3-Gallon Jug: (3,0) --> (3,3) |

| Pour water from 3-Gallon jug to 4-gallon until it is full: (3,3) --> (4,2) |

| Empty the 4-Gallon jug on ground: (4,2) --> (0,2) |

| Pour all the water from 3-Gallon jug to 4-gallon: (0,2) --> (2,0) |

| |

| Press the SPACE bar |

| |

| |

| |

+----------------------------------------------------------------------------+

1. Write simple Prolog functions such as the following. Take into account lists which are too short.

-- remove the Nth item from the list. -- insert as the Nth item.

SOLUTION:

<br />

<b>Warning</b>: include(ai/week9.pl): failed to open stream: No such file or directory in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>113</b><br />

<br />

<b>Warning</b>: include(): Failed opening 'ai/week9.pl' for inclusion (include\_path='.:/opt/alt/php73/usr/share/pear') in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>113</b><br />

OUTPUT:

<br />

<b>Warning</b>: include(ai/outputs/w9.txt): failed to open stream: No such file or directory in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>165</b><br />

<br />

<b>Warning</b>: include(): Failed opening 'ai/outputs/w9.txt' for inclusion (include\_path='.:/opt/alt/php73/usr/share/pear') in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>165</b><br />

1. Assume the prolog predicate gt(A, B) is true when A is greater than B. Use this predicate to define the predicate addLeaf(Tree, X, NewTree) which is true if NewTree is the Tree produced by adding the item X in a leaf node. Tree and NewTree are binary search trees. The empty tree is represented by the atom nil.

SOLUTION:

\ <br />

<b>Warning</b>: include(ai/week10.pl): failed to open stream: No such file or directory in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>113</b><br />

<br />

<b>Warning</b>: include(): Failed opening 'ai/week10.pl' for inclusion (include\_path='.:/opt/alt/php73/usr/share/pear') in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>113</b><br />

OUTPUT:

<br />

<b>Warning</b>: include(ai/outputs/w10.txt): failed to open stream: No such file or directory in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>165</b><br />

<br />

<b>Warning</b>: include(): Failed opening 'ai/outputs/w10.txt' for inclusion (include\_path='.:/opt/alt/php73/usr/share/pear') in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>165</b><br />

1. Write a Prolog predicate, countLists(Alist, Ne, Nl), using accumulators, that is true when Nl is the number of items that are listed at the top level of Alist and Ne is the number of empty lists. Suggestion: First try to count the lists, or empty lists, then modify by adding the other counter.

**SOLUTION:**

<br />

<b>Warning</b>: include(ai/week11.pl): failed to open stream: No such file or directory in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>113</b><br />

<br />

<b>Warning</b>: include(): Failed opening 'ai/week11.pl' for inclusion (include\_path='.:/opt/alt/php73/usr/share/pear') in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>113</b><br />

**OUTPUT:**

<br />

<b>Warning</b>: include(ai/outputs/w11.txt): failed to open stream: No such file or directory in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>165</b><br />

<br />

<b>Warning</b>: include(): Failed opening 'ai/outputs/w11.txt' for inclusion (include\_path='.:/opt/alt/php73/usr/share/pear') in <b>/home/u681245571/domains/studyglance.in/public\_html/labprograms/sdcdisplay.php</b> on line <b>165</b><br />

1. Define a predicate memCount(AList,Blist,Count) that is true if Alist occurs Count times within Blist. Define without using an accumulator. Use "not" as defined in utilities.pro, to make similar cases are unique, or else you may get more than one count as an answer.

Examples:

memCount(a,[b,a],N). N = 1 ;

no memCount(a,[b,[a,a,[a],c],a],N). N = 4 ;

no memCount([a],[b,[a,a,[a],c],a],N). N = 1 ;

No