AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY

LABORATORY MANUAL

ARTIFICIAL INTELLIGENCE

SUBJECT CODE: 2180703

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

B.E. 8th SEMESTER

NAME:_____
ENROLLMENT NO: _____
BATCH NO: _____
YEAR: _____

Amiraj College of Engineering and Technology,

Nr.Tata Nano Plant, Khoraj, Sanand, Ahmedabad.

AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY

Amiraj College of Engineering and Technology,

Nr.Tata Nano Plant, Khoraj, Sanand, Ahmedabad.

CERTIFICATE

This is to certify that Mr. / Ms. _____

Of class_____ Enrolment No _____has

Satisfactorily completed the course in _____as

by the Gujarat Technological University for ____ Year (B.E.) semester___ of

Computer Science and Engineering in the Academic year_____.

Date of Submission:-

Faculty Name and Signature

(Subject Teacher)

Head of Department

(Computer)

AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

B.E. 8th SEMESTER

SUBJECT: ARTIFICIAL INTELLIGENCE

SUBJECT CODE: 2180703

List Of Experiments

Sr. No.	Title	Date of Performance	Date of submission	Sign	Remark
1.	Write a program to implement Tic-Tac-Toe game problem.				
2	Write a PROLOG/C program to solve Water Jug Problem (Show the solution path also).				
3	Write a PROLOG/C program to solve the N-Queen Problem.				
4	Write a PROLOG/C program to implement 8 – puzzle problem using Depth First Search.				
5	Write a PROLOG/C program to implement 8 – puzzle problem using A* Algorithm.				
6	Write a program to solve 8-Queens problem using Prolog.				

7	A PROLOG/C program to implement an Expert System of your choice.				
8	Write a PROLOG/C program to solve Monkey-Banana problem.				
9	Write a PROLOG/C program to solve the Travelling Salesman Problem.				
10	Write a PROLOG Program to generate query based on family relationship.				

PRACTICAL-1

Aim:- Write a program to implement Tic-Tac-Toe game problem.

Program:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i = 0;                      /* Loop counter                    */
        int player = 0;                 /* Player number - 1 or 2          */
        int go = 0;                     /* Square selection number for turn  */
        int row = 0;                    /* Row index for a square          */
        int column = 0;                 /* Column index for a square       */
        int line = 0;                   /* Row or column index in checking loop */
        int winner = 0;                 /* The winning player              */
        char board[3][3] =
        {                               /* The board                       */
        {'1','2','3'},                  /* Initial values are reference numbers */
        {'4','5','6'},                  /* used to select a vacant square for  */
        {'7','8','9'}                   /* a turn.                         */
        };
        clrscr();
        /* The main game loop. The game continues for up to 9 turns */
        /* As long as there is no winner                            */
        for( i = 0; i<9 && winner==0; i++)
        {
                /* Display the board */
                printf("\n\n");
                printf(" %c | %c | %c\n", board[0][0], board[0][1], board[0][2]);
                printf("---+---+---\n");
                printf(" %c | %c | %c\n", board[1][0], board[1][1], board[1][2]);
                printf("---+---+---\n");
                printf(" %c | %c | %c\n", board[2][0], board[2][1], board[2][2]);
                player = i%2 + 1;               /* Select player */
                /* Get valid player square selection */
                do
```

```
            {
                printf("\nPlayer %d, please enter the number of the square "
                "where you want to place your %c: ", player,(player==1)?'X':'O');
                scanf("%d", &go);
                row = --go/3;                   /* Get row index of square      */
                column = go%3;                  /* Get column index of square   */
            }
            while(go<0 || go>9 || board[row][column]>'9');
            board[row][column] = (player == 1) ? 'X' : 'O';       /* Insert player symbol   */
            /* Check for a winning line - diagonals first */
            if((board[0][0] == board[1][1] && board[0][0] == board[2][2]) ||
            (board[0][2] == board[1][1] && board[0][2] == board[2][0]))
            winner = player;
            else
            /* Check rows and columns for a winning line */
            for(line = 0; line <= 2; line ++)
            if((board[line][0] == board[line][1] && board[line][0] == board[line][2])||
            (board[0][line] == board[1][line] && board[0][line] == board[2][line]))
            winner = player;
        }
/* Game is over so display the final board */
        printf("\n\n");
        printf(" %c | %c | %c\n", board[0][0], board[0][1], board[0][2]);
        printf("---+---+---\n");
        printf(" %c | %c | %c\n", board[1][0], board[1][1], board[1][2]);
        printf("---+---+---\n");
        printf(" %c | %c | %c\n", board[2][0], board[2][1], board[2][2]);
        /* Display result message */
        if(winner == 0)
        printf("\nHow boring, it is a draw\n");
        else
        printf("\nCongratulations, player %d, YOU ARE THE WINNER!\n", winner);
        getch():
}
```

Output:-

```
 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9

Player 1, please enter the number of the square where you want to place your X:
5


 1 | 2 | 3
---+---+---
 4 | X | 6
---+---+---
 7 | 8 | 9

Player 2, please enter the number of the square where you want to place your O:
_
```

```
Player 2, please enter the number of the square where you want to place your O:
4


 X | O | O
---+---+---
 O | X | 6
---+---+---
 X | X | O

Player 1, please enter the number of the square where you want to place your X:
6


 X | O | O
---+---+---
 O | X | X
---+---+---
 X | X | O

How boring, it is a draw
```

PRACTICAL-2

Aim:- Write a PROLOG/C program to solve Water Jug Problem (Show the solution path also).

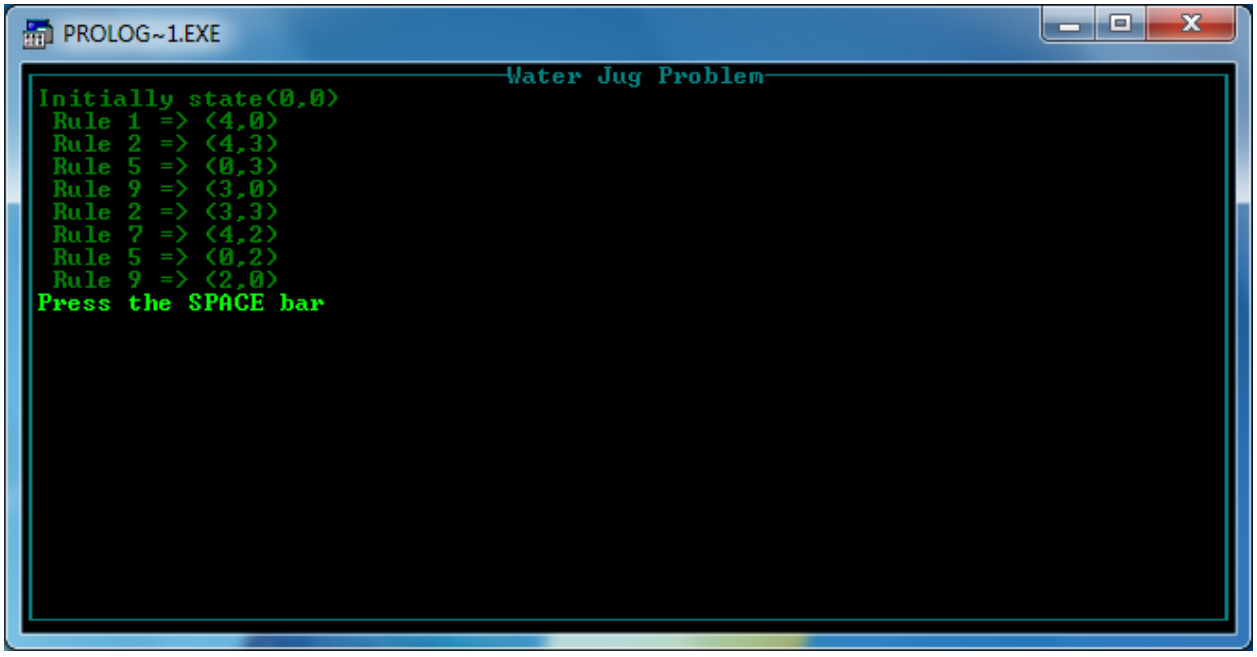
Program:-

```
database
   rstate(integer,integer)
predicates
   state(integer,integer)

clauses

   state(2,_).
   state(0,0):-
      not(rstate(0,0)),
      assert(rstate(0,0)),
      state(0,0).

   state(X,Y):-

      X < 4,
      not(rstate(4,Y)),
      assert(rstate(4,Y)),
      write("\n Rule 1 => (4,",Y,")"),
      state(4,Y).

   state(X,Y):-

      Y < 3,
      not(rstate(X,3)),
      assert(rstate(X,3)),
      write("\n Rule 2 => (",X,",3)"),
      state(X,3).

   state(X,Y):-

      X>0,
      not(rstate(0,Y)),
      assert(rstate(0,Y)),
      write("\n Rule 5 => (0,",Y,")"),
      state(0,Y).

   state(X,Y):-
```

```
    Y>0,
    not(rstate(X,0)),
    assert(rstate(X,0)),
    write("\n Rule 6 => (",X,",0)"),
    state(X,0).

state(X,Y):-

    X+Y >= 4,
    Y > 0,
    Z=Y-(4-X),
    not(rstate(4,Z)),
    assert(rstate(4,Z)),
    write("\n Rule 7 => (4,",Z,")"),
    state(4,Z).

state(X,Y):-

    X+Y >= 3,
    X>0,
    Z=X-(3-Y),
    not(rstate(Z,3)),
    assert(rstate(Z,3)),
    write("\n Rule 8 => (",Z,",3)"),
    state(Z,3).

state(X,Y):-

    X+Y <= 4,
    Y > 0,
    Z=X+Y,
    not(rstate(Z,0)),
    assert(rstate(Z,0)),
    write("\n Rule 9 => (",Z,",0)"),
    state(Z,0).

state(X,Y):-

    X+Y <= 3,
    X>0,
    Z=X+Y,
    not(rstate(0,Z)),
    assert(rstate(0,Z)),
    write("\n Rule 10 => (0,",Z,")"),
    state(0,Z).
```

```
state(X,Y):-

    X=0,
    Y=2,
    not(rstate(2,0)),
    assert(rstate(2,0)),
    write("\n Rule 11 => (2,0)"),
    state(2,0).

state(X,Y):-

    X=2,
    assert(rstate(0,Y)),
    write("\n Rule 12 => (0,",Y,")"),
    state(0,Y).

goal
   makewindow(1,2,3,"Water Jug Problem",0,0,25,80),
   write("Initially state(0,0)"),
   state(0,0).
```

Output:-



PRACTICAL-3

Aim:- Write a PROLOG/C program to solve the N-Queen Problem.

Program:-

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a[30],count=0;
int place(int pos)
{
	int i;
	for(i=1;i<pos;i++)
	{
		if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
		return 0;
	}
	return 1;
}
void print_sol(int n)
{
	int i,j;
	count++;
	printf("\n\nSolution #%d:\n",count);
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			if(a[i]==j)
			printf("Q\t");
			else
			printf("*\t");
		}
		printf("\n");
	}
}
void queen(int n)
{
	int k=1;
	a[k]=0;
	while(k!=0)
	{
		a[k]=a[k]+1;
		while((a[k]<=n)&&!place(k))
```

```
                a[k]++;
                if(a[k]<=n)
                {
                        if(k==n)
                        print_sol(n);
                        else
                        {
                                k++;
                                a[k]=0;
                        }
                }
                else
                k--;
        }
}
void main()
{
        int i,n;
        clrscr();
        printf("Enter the number of Queens\n");
        scanf("%d",&n);
        queen(n);
        printf("\nTotal solutions=%d",count);
        getch();
}
```

Output:-

```
Enter the number of Queens
4


Solution #1:
*       Q       *       *
*       *       *       Q
Q       *       *       *
*       *       Q       *


Solution #2:
*       *       Q       *
Q       *       *       *
*       *       *       Q
*       Q       *       *

Total solutions=2
```

PRACTICAL-4

Aim:-Write a PROLOG/C program to implement 8 – puzzle problem using Depth First Search.

Program:-

```
domains
   value, row, col, gval, hval, pval, sval, parent, nodeno = integer
   nodevalue=ndval(value,row,col)
   nodelist=nodevalue*
   loclist=value*
   poslist=posval(loclist,value)
   nodestruct=ndstruct(nodelist,value,value,value,value)
   hvallist=nodestruct*

database
   opennodeinfo(nodelist,value,value,value,value).
   closenodeinfo(nodelist,value,value,value,value).
   bestnodeinfo(nodelist,value,value,value,value).
   nvalue(value,row,col).
   rowcolCounter(value).
   nodeNo(value).
   currParent(value).

predicates
   displayPuzzle.
   displayNodeList(nodelist).
   calHvalue(nodelist,value).
   calPvalue(nodelist,value).
   getNodeInfo(nodevalue,value,row,col).
   finalnode(nodelist,hval).
   pvalue(value, poslist).
   findStepsFar(value,poslist,value).
   memberOfLocList(value,loclist).
   moveLeft.
   moveRight.
   moveUp.
   moveDown.
   findValue(row,col,value).
   findBlank(row,col).
   setNewPos(value,row,col).
   setCurrNode(nodelist).
   emptyCurrNode.
   genNodeList(nodelist,nodelist).
   genHvalList(hvallist,hvallist).
```

```
    input.
    performSearch.
    processCurrNode.
    checkInOpen(nodelist,value).
    checkInClose(nodelist,value).
    bubblesort(hvallist,hvallist).
    swap(hvallist,hvallist).
    insertSortedList(hvallist).

clauses
    finalnode([ndval(1,0,0),ndval(2,0,1),ndval(3,0,2),ndval(8,1,0),
        ndval(0,1,1),ndval(4,1,2),ndval(7,2,0),ndval(6,2,1),
        ndval(5,2,2)],0).

    pvalue(1,posval([0],0)).
    pvalue(1,posval([1,3],1)).
    pvalue(1,posval([2,6,4],2)).
    pvalue(1,posval([7,5],3)).
    pvalue(1,posval([8],4)).

    pvalue(2,posval([1],0)).
    pvalue(2,posval([0,2,4],1)).
    pvalue(2,posval([3,5,7],2)).
    pvalue(2,posval([6,8],3)).

    pvalue(3,posval([2],0)).
    pvalue(3,posval([1,5],1)).
    pvalue(3,posval([0,4,8],2)).
    pvalue(3,posval([3,7],3)).
    pvalue(3,posval([6],4)).

    pvalue(4,posval([5],0)).
    pvalue(4,posval([2,4,8],1)).
    pvalue(4,posval([1,3,7],2)).
    pvalue(4,posval([0,6],3)).

    pvalue(5,posval([8],0)).
    pvalue(5,posval([5,7],1)).
    pvalue(5,posval([2,4,6],2)).
    pvalue(5,posval([1,3],3)).
    pvalue(5,posval([0],4)).

    pvalue(6,posval([7],0)).
    pvalue(6,posval([4,6,8],1)).
    pvalue(6,posval([1,3,5],2)).
    pvalue(6,posval([0,2],3)).
```

```
pvalue(7,posval([6],0)).
pvalue(7,posval([3,7],1)).
pvalue(7,posval([0,4,8],2)).
pvalue(7,posval([1,5],3)).
pvalue(7,posval([2],4)).

pvalue(8,posval([3],0)).
pvalue(8,posval([0,4,6],1)).
pvalue(8,posval([1,5,7],2)).
pvalue(8,posval([2,8],3)).

pvalue(0,posval([4],0)).
pvalue(0,posval([1,3,5,7],1)).
pvalue(0,posval([0,2,6,8],2)).

input:-
   assert(opennodeinfo([ndval(2,0,0),ndval(1,0,1),
      ndval(6,0,2),ndval(4,1,0),ndval(0,1,1),
      ndval(8,1,2),ndval(7,2,0),ndval(5,2,1),
      ndval(3,2,2)],0,0,0,0)),
   assert(rowcolCounter(8)),
   assert(nodeNo(1)),
   assert(currParent(0)).

displayPuzzle:-
   genNodeList([],Nodelist),
   bestnodeinfo(Nodelist,Hval,Gval,Parent,Nodeno),
   displayNodeList(Nodelist),
   write("h(n) : ",Hval),nl,
   write("g(n) : ",Gval),nl,
   write("parent(n) : ",Parent),nl,
   write("nodno(n) : ",Nodeno).

displayNodeList([]).
displayNodeList([ndval(Value1,_,_),ndval(Value2,_,_),ndval(Value3,_,_)|Tail]):-
   write(Value1," "), write(Value2," "), write(Value3," "),nl,
   displayNodeList(Tail).

getNodeInfo(ndval(Value,Row,Col),Value,Row,Col).

emptyCurrNode:-
   retractall(nvalue(_,_,_)).

setCurrNode([]):- !.
setCurrNode([ndval(Value,Row,Col)|Tail]):-
```

```
    assert(nvalue(Value,Row,Col)),
    setCurrNode(Tail).

calHvalue(NodeList,Hval):-
    calPvalue(NodeList,Pval),
    Hval=Pval.

calPvalue([],0):- !.
calPvalue([Head|Tail],Pval):-
    getNodeInfo(Head,Value,Row,Col),
    CurrPos=(Row*3)+Col,
    pvalue(Value,PosList),
    findStepsFar(CurrPos,PosList,Steps),
    calPvalue(Tail,NewPval),
    Pval=Steps+NewPval.

findStepsFar(CurrPos,posval(LocList,Steps),Steps):-
    memberOfLocList(CurrPos,LocList).

memberOfLocList(CurrPos,[CurrPos|_]) :- !.
memberOfLocList(CurrPos,[_|Rest]):-
    memberOfLocList(CurrPos,Rest).

moveLeft:-
    findBlank(Row,Col),
    Col>0,
    NewCol=Col-1,
    findValue(Row,NewCol,Value),
    setNewPos(Value,Row,Col),
    setNewPos(0,Row,NewCol),
    write("\nleft successor"),
    processCurrNode,
    setNewPos(0,Row,Col),
    setNewPos(Value,Row,NewCol),!.
moveLeft.

moveRight:-
    findBlank(Row,Col),
    Col<2,
    NewCol=Col+1,
    findValue(Row,NewCol,Value),!,
    setNewPos(Value,Row,Col),
    setNewPos(0,Row,NewCol),
    write("\nright successor"),
    processCurrNode,
    setNewPos(0,Row,Col),
```

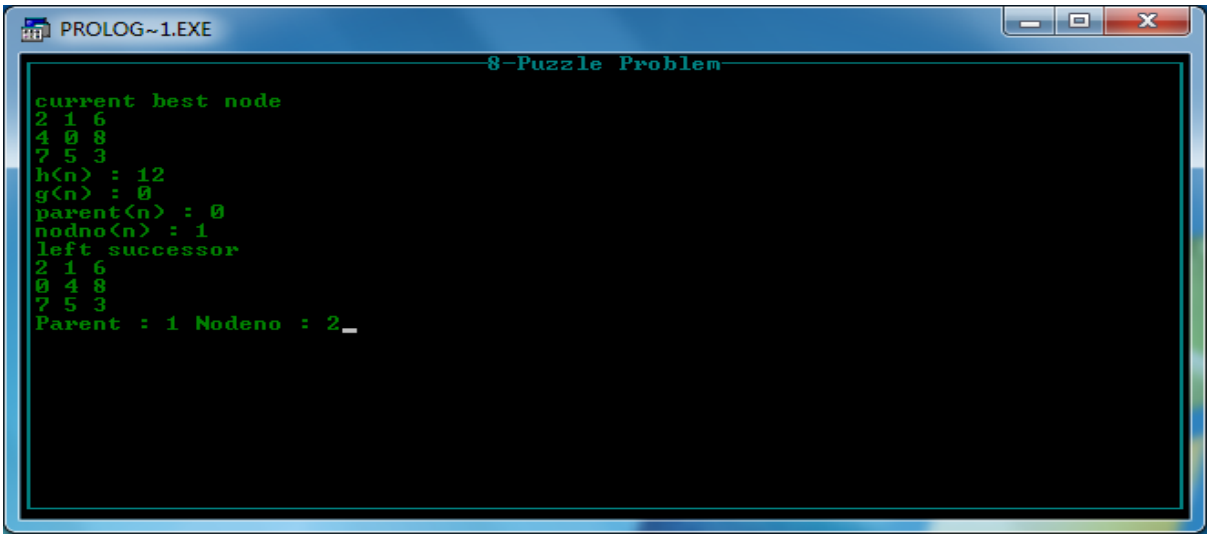
```
    setNewPos(Value,Row,NewCol),!.
  moveRight.

  moveUp:-
    findBlank(Row,Col),
    Row>0,
    NewRow=Row-1,
    findValue(NewRow,Col,Value),
    setNewPos(Value,Row,Col),
    setNewPos(0,NewRow,Col),
    write("\nup successor"),
    processCurrNode,
    setNewPos(0,Row,Col),
    setNewPos(Value,NewRow,Col),!.
  moveUp.
  moveDown:-
    findBlank(Row,Col),
    Row<2,
    NewRow=Row+1,
    findValue(NewRow,Col,Value),
    setNewPos(Value,Row,Col),
    SetNewPos(0,NewRow,Col),
    write("\ndown successor"),
    processCurrNode,
    setNewPos(0,Row,Col),
    setNewPos(Value,NewRow,Col),!.
  moveDown.

  processCurrNode:-
    genNodeList([],Nodelist),
    checkInOpen(Nodelist,Oval),
    Oval=1,
    checkInClose(Nodelist,Cval),
    Cval=1,
    calHvalue(Nodelist,Hval),
    currParent(Parent),
    Gval=Parent+1,
    nodeNo(Nodeno),
    NewNodeno=Nodeno+1,
    retract(nodeNo(Nodeno)),
    assert(nodeNo(NewNodeno)),
    assert(opennodeinfo(Nodelist,Hval,Gval,Parent,Nodeno)),
    nl,displayNodeList(Nodelist), write("Parent : ",Parent," Nodeno : ",Nodeno),
    readchar(_),
    genHvalList([],HvalList),
    bubblesort(HvalList,SortedList),
```

```
    insertSortedList(SortedList).
  processCurrNode.

  checkInOpen(Nodelist,Oval):-
    Oval=1,
    not(opennodeinfo(Nodelist,_,_,_,_)),!.
  checkInOpen(Nodelist,Oval):-
    opennodeinfo(Nodelist,_,_,_,_),
    Oval=0.

  checkInClose(Nodelist,Cval):-
    Cval=1,
    not(closenodeinfo(Nodelist,_,_,_,_)),!.
  checkInClose(Nodelist,Cval):-
    closenodeinfo(Nodelist,_,_,_,_),
    Cval=0.
    findValue(Row,Col,Value):-
    nvalue(Value,Row,Col).
 findBlank(Row,Col):-
    nvalue(0,Row,Col).

  setNewPos(Value,Row,Col):-
    retract(nvalue(_,Row,Col)),
    assert(nvalue(Value,Row,Col)).

  genNodeList(L,NodeList):-
    rowcolCounter(Cnt),
    Cnt>=0,
    NewCnt=Cnt-1,
    retract(rowcolCounter(Cnt)),
    assert(rowcolCounter(NewCnt)),
    Row=Cnt div 3,
    Col=Cnt mod 3,
    nvalue(Value,Row,Col),
    NewList=[ ndval(Value,Row,Col) | L ],!,
    genNodeList(NewList,NodeList).
  genNodeList(NodeList,NodeList):-
    rowcolCounter(Cnt),
    retract(rowcolCounter(Cnt)),
    assert(rowcolCounter(8)),!.

  genHvalList(L,Hvallist):-
    retract(opennodeinfo(Nodelist,Hval,Gval,Parent,Nodeno)),
    Newlist=[ndstruct(Nodelist,Hval,Gval,Parent,Nodeno)|L],
    genHvalList(Newlist,Hvallist).
  genHvalList(Hvallist,Hvallist):- !.
```

```
   insertSortedList([ndstruct(Nodelist,Hval,Gval,Parent,Nodeno)|Tail]):-
      assert(opennodeinfo(Nodelist,Hval,Gval,Parent,Nodeno)),
      insertSortedList(Tail).
   insertSortedList([]) :- !.

   performSearch:-
      retract(bestnodeinfo(_,_,_,_,_)),
      opennodeinfo(Nodelist,Hval,Gval,Parent,Nodeno),
      Hval<> 0,
      retract(currParent(_)),
      assert(currParent(Nodeno)),
      assert(bestnodeinfo(Nodelist,Hval,Gval,Parent,Nodeno)),
      emptyCurrNode,
      setCurrNode(Nodelist),
      write("\ncurrent best node\n"),
      displayNodeList(Nodelist),
      write("h(n) : ",Hval),nl,
      write("g(n) : ",Gval),nl,
      write("parent(n) : ",Parent),nl,
      write("nodno(n) : ",Nodeno),
      readchar(_),
      assert(closenodeinfo(Nodelist,Hval,Gval,Parent,Nodeno)),
      retract(opennodeinfo(Nodelist,Hval,Gval,Parent,Nodeno)),
      moveLeft,
      moveUp,
      moveRight,
      moveDown,
      performSearch.
   performSearch.

bubblesort(List, Sorted) :-
    swap(List, List1), !,
    bubblesort(List1, Sorted).

bubblesort(Sorted, Sorted).

swap([ndstruct(A,X,B,C,D),ndstruct(E,Y,F,G,H)|Rest], [ndstruct(E,Y,F,G,H),ndstruct(A,X,B,C,D)|Rest]) :- X >
Y.
swap([ndstruct(A,Z,B,C,D)|Rest], [ndstruct(A,Z,B,C,D)|Rest1]) :- swap(Rest, Rest1).

goal
   makewindow(1,2,3,"8-Puzzle Problem",0,0,25,80),
   input,
   opennodeinfo(Nodelist,_,_,_,_),
   calHvalue(Nodelist,Hval),
```

```
nodeNo(Nodeno),
NewNodeno=Nodeno+1,
retract(nodeNo(Nodeno)),
assert(nodeNo(NewNodeno)),
retract(opennodeinfo(Nodelist,_,_,_,_)),
assert(opennodeinfo(Nodelist,Hval,0,0,Nodeno)),
assert(bestnodeinfo(Nodelist,Hval,0,0,Nodeno)),
setCurrNode(Nodelist),
emptyCurrNode,
performSearch.
```

Output:-



PRACTICAL-5

Aim:- Write a PROLOG/C program to implement 8 – puzzle problem using A* Algorithm.

Program:-

```
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <conio.h>

#define DOWN 0
#define UP 1
#define LEFT 2
#define RIGHT 3
#define H2

struct elementstruct
{
        int block[9];
        char* str;
        int pathcost;
        int valid;
        int totalcost;
        elementstruct* next;
};

int heur(int block[]);
void prepend(elementstruct* newnode, elementstruct* oldnode, int operator1);
int goal(int* block);
int notonqueue(int block[]);
elementstruct* bestnodefromqueue();
void print_block(int* block);
int apply (int* newstate, int* oldstate, int op);
elementstruct* newelement();
int op(char);
char to_char(int i);

char rep[] = "dulr";
int notvalid1[4] = { 6, 0, 0, 2 };
int notvalid2[4] = { 7, 1, 3, 5 };
int notvalid3[4] = { 8, 2, 6, 8 };
int applyparam[4] = { +3, -3, -1, +1 };
```

```
int goal_block[9] = { 0, 1, 2, 3, 4, 5, 6, 7, 8}; //8 indicates no tile
int maxdepth;
elementstruct* top;

int main()
{
        int block[9];
        clrscr();
        printf("\nThe Eight Puzzle!\n");
        printf("\nPlease Enter the initial state of the game \n"
            " [Represent tiles with numbers 1 to 8, and the blank space as 'x'.\n"
          " Start writing them from left to right for each row. Start from the topmost row to the   bottommost
                                                                row.\n"
            " Your final string will look similar to this '1 4 2 3 x 6 7 8 5'.\n"
            " Do not forget the spaces in between the characters]\n");
        int i = 0;
        while(i<9)
                {
                        char chr;
                        chr = fgetc(stdin);
                        if (chr==32) continue;
                        if (chr=='x') block[i] = 8;
                        else if (chr >= '1' && chr <= '9') block[i] = chr - '1';
                        else { printf("Invalid Input. Example of valid input...2 1 3 4 7 5 6 8 x.", chr); return 1;
                }
                i++;
}

 fgetc(stdin); //flush out the end of line character
 printf("\n Now Enter the Goal State in a similar way. (Typical. 1 2 3 4 5 6 7 8 x): ");
 i = 0;
 while(i<9)
        {
                char chr;
                chr = fgetc(stdin);
                if (chr==32) continue;
                if (chr=='x') goal_block[i] = 8;
                else if (chr >= '1' && chr <= '9') goal_block[i] = chr - '1';
                else { printf("chr=%d. Invalid Input. Example of valid input...2 1 3 4 7 5 6 8 x.",(int) chr); return
                1; }
                i++;
        }
          printf("Enter the maximum depth you want to search (<25 is solved quickly): ");
          scanf("%d", &maxdepth);

        printf("\nWorking...");
```

```
top = newelement();
for(i=0; i<9; i++)
top->block[i] = block[i];
top->totalcost = heur(block);
elementstruct* newnode = newelement();
while (1)
{
        elementstruct* node = bestnodefromqueue();
        if (node == NULL)
        {
                printf("done!\n");
                printf("There is no solution to this of less than %d depth.\n", maxdepth);
                printf("Try increasing the depth by 5.\n");
                printf("If there is no solution within 35-40 depth, the pattern is usually unsolvable.\n\n");
                break;
        }
        else if (goal(node->block))
        {
                char chr[15];
                printf("done. \nFound the solution of least number of steps (%d).", node->pathcost);
                printf("\nWant a graphical display of each step? (Y/N)?");
                scanf("%s", chr);
                if(chr[0] =='n' || chr[0]=='N') {
                printf("\n (Move Blank u=up, d=down, l=left, r=right)\n");
                printf(node->str);
                printf("\n");
                break;
        }
        int block2[9];
        for (i=0; i<node->pathcost; i++)
        {
                print_block(block);
                apply(block2, block, op(node->str[i]));
                for(int j=0; j<=8; j++)
                block[j] = block2[j];
        }
        print_block(block);
        printf("\nGraphical Display Complete.\nThe steps taken were: (Move blank u=up, d=down, l-left,
        r=right)\n");
        printf(node->str);
        printf("\n");
        break;
}
if (node->totalcost > maxdepth) continue;
for(i=0; i<=3; i++)
{
```

```
                if (apply(newnode->block, node->block, i) == -1)
                continue;
                if (notonqueue(newnode->block))
                {
                        prepend(newnode, node, i);
                        newnode = newelement();
                        if (newnode==NULL) { printf ("ERROR!! insufficient memory!! Try decreasing depth!");
                        return 1;
                }
        }
    }
}
return 0;
}

int heur(int* block)
{
        #ifdef H2
        int to_return = 0;
        for(int i=0; i<9; i++)
        {
                to_return += abs((i/3) - (block[i]/3));
                to_return += abs((i%3) - (block[i]%3));
        }
        return to_return;
        #else
        int to_0return = 0;
        for(int i=0; i<9; i++)
        {
                if (block[i] != i) to_return++;
        }
        return to_return;
        #endif
}
void prepend(elementstruct* newnode, elementstruct* oldnode, int op)
{
        newnode->next = top;
        top = newnode;
        strcpy(newnode->str, oldnode->str);
        newnode->str[oldnode->pathcost] = rep[op];
        newnode->str[oldnode->pathcost+1] = 0;
        newnode->pathcost = oldnode->pathcost+1;
        newnode->totalcost = newnode->pathcost + heur(newnode->block);
        if (newnode->totalcost < oldnode->totalcost) newnode->totalcost = oldnode->totalcost;
}
int goal(int* block)
```

```
{
        int* g_block = goal_block;
        for(int i=0; i<9; i++)
        if ((*(block++))!=(*(g_block++)))
        return 0;
        return 1;
}
int notonqueue(int* block)
{
        int i,j;
        elementstruct* t = top;
        while (t!=NULL)
        {
                for(i=0; i<9; i++)
                if (t->block[i] != block[i]) break;
                if (i==9) return 0;
                t = t->next;
        }
        return 1;
}
elementstruct* bestnodefromqueue()
{
          elementstruct* t = top;
          int min_totalpathcost = 1000;
          int totalpathcost;
          elementstruct* to_return = NULL;

          while (t != NULL)
         {
                if (t->valid==1 && t->totalcost < min_totalpathcost)
                {
                        min_totalpathcost = t->totalcost;
                        to_return = t;
                }
                t = t->next;
        }
         if (to_return != NULL) to_return->valid = 0;
        return to_return;
}
int apply (int* newstate, int* oldstate, int op)
{
        int j;
        int blank;
        for (j=0; j<9; j++)
        if (oldstate[j]==8) { blank=j; break; }
        if (blank==notvalid1[op] || blank==notvalid2[op] || blank==notvalid3[op])
```

```
        return -1;
        for (j=0; j<9; j++)
        newstate[j] = oldstate[j];
        newstate[blank] = newstate[blank+applyparam[op]];
        newstate[blank+applyparam[op]] = 8;
        return 1;
}
elementstruct* newelement()
{
        elementstruct* t = new elementstruct;
        if (t==NULL) return NULL;
        t->valid = 1;
        t->str = new char[maxdepth+1];
        if (t->str ==NULL) return NULL;
        t->str[0] = 0;
        t->pathcost = t->totalcost = 0;
        t->next = NULL;
        return t;
}
void print_block(int* block)
{
        printf("\n");
        printf ("\n-------");
        printf ("\n|%c|%c|%c|", to_char(block[0]), to_char(block[1]), to_char(block[2]));
        printf ("\n-------");
        printf ("\n|%c|%c|%c|", to_char(block[3]), to_char(block[4]), to_char(block[5]));
        printf ("\n-------");
        printf ("\n|%c|%c|%c|", to_char(block[6]), to_char(block[7]), to_char(block[8]));
        printf ("\n-------");
}

char to_char(int i)
{
          if (i>=0 &&i<=7) return i+'1';
          else if (i==8) return 'x';
          else
          {
                printf("ERROR in Program!"); return -1;
           }
}
int op(char i)
{
        switch (i)
        {
                  case 'd': return 0;
                  case 'u': return 1;
```

```
                case 'l': return 2;
                case 'r': return 3;
                default: printf("ERROR!"); return -1;
        }
}
```

Output:-

```
The Eight Puzzle!

Please Enter the initial state of the game
 [Represent tiles with numbers 1 to 8, and the blank space as 'x'.
 Start writing them from left to right for each row. Start from the topmost row
to the bottommost row.
 Your final string will look similar to this '1 4 2 3 x 6 7 8 5'.
 Do not forget the spaces in between the characters]
1 2 3 8 x 4 7 6 5

 Now Enter the Goal State in a similar way. (Typical. 1 2 3 4 5 6 7 8 x): 1 2 3
x 8 4 7 6 5
Enter the maximum depth you want to search (<25 is solved quickly): 10

Working...done.
Found the solution of least number of steps (1).
Want a graphical display of each step? (Y/N)?
```

```
Working...done.
Found the solution of least number of steps (1).
Want a graphical display of each step? (Y/N)?y


-------
¦1¦2¦3¦
-------
¦8¦x¦4¦
-------
¦7¦6¦5¦
-------

-------
¦1¦2¦3¦
-------
¦x¦8¦4¦
-------
¦7¦6¦5¦
-------
Graphical Display Complete.
The steps taken were: (Move blank u=up, d=down, l-left, r=right)
l
```

PRACTICAL-6

Aim:- Write a program to solve 8-Queens problem using Prolog.
Program:-

P= {(x1, y1),(x2,y2),……..(x8,y8)}
where (x1,y1) gives the position of the first queen and so on. So it can be clearly seen that the domains for xi and yi are
Dx = {1,2,3,4,5,6,7,8}and Dy ={1,2,3,4,5,6,7,8} respectively.
The constraints are
i. No two queens should be in the same row,
i.e yi≠yj for i=1 to 8;j=1 to 8;i≠j
ii. No two queens should be in the same column,
i.e xi≠xj for i=1 to 8;j=1 to 8;i≠j
iii. There should not be two queens placed on the same diagonal line
i.e (yi-yj) ≠ ±(xi-xj).

```
DOMAINS
cell=c(integer,integer)
list=cell*
int_list=integer*
PREDICATES
solution(list)
member(integer,int_list)
nonattack(cell,list)
CLAUSES
solution([]).
solution([c(X,Y)|Others]):-
solution(Others),
member(Y,[1,2,3,4,5,6,7,8]),
nonattack(c(X,Y),Others).
nonattack(_,[]).
nonattack(c(X,Y),[c(X1,Y1)|Others]):-
Y<>Y1,
Y1-Y<>X1-X,
Y1-Y<>X-X1,
nonattack(c(X,Y),Others).
member(X,[X|_]).
member(X,[_|Z]):-
member(X,Z).
GOAL
solution([c(1,A),c(2,B),c(3,C),c(4,D),c(5,E),c(6,F),c(7,G),c(8,H)]).
```

Solution:
A=4, B=2, C=7, D=3, E=6, F=8, G=5, H=1
A=5, B=2, C=4, D=7, E=3, F=8, G=6, H=1
A=3, B=5, C=2, D=8, E=6, F=4, G=7, H=1
A=3, B=6, C=4, D=2, E=8, F=5, G=7, H=1
A=5, B=7, C=1, D=3, E=8, F=6, G=4, H=2

Total 92 solutions

PRACTICAL-7

Aim:- Write a PROLOG/C program to implement an Expert System of your choice.

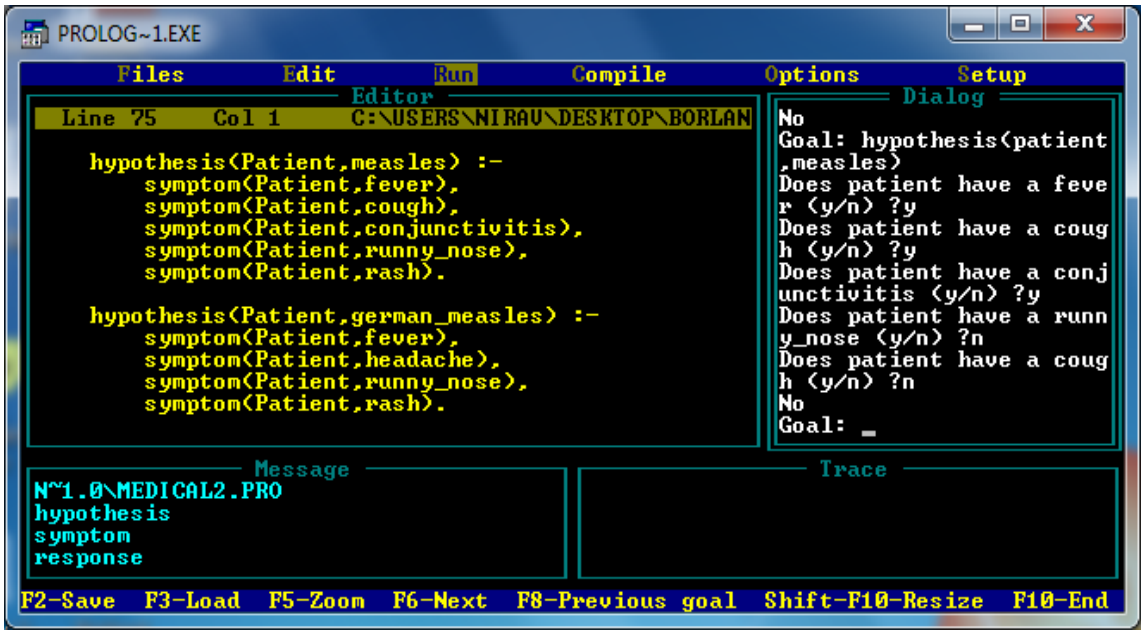
Program:-

```
domains
   disease,indication = symbol
   Patient,name = string

predicates
   hypothesis(string,disease)
   symptom(name,indication)
   response(char)
   go
clauses
   go :-
      write("What is the patient's name? "),
      readln(Patient),
      hypothesis(Patient,Disease),
      write(Patient,"probably has ",Disease,"."),nl.

   go :-
      write("Sorry, I don't seem to be able to"),nl,
      write("diagnose the disease."),nl.

   symptom(Patient,fever) :-
      write("Does ",Patient," have a fever (y/n) ?"),
      response(Reply),
      Reply='y'.

   symptom(Patient,rash) :-
      write("Does ",Patient," have a rash (y/n) ?"),
      response(Reply),
      Reply='y'.

   symptom(Patient,headache) :-
      write("Does ",Patient," have a headache (y/n) ?"),
      response(Reply),
      Reply='y'.

   symptom(Patient,runny_nose) :-
      write("Does ",Patient," have a runny_nose (y/n) ?"),
      response(Reply),
      Reply='y'.
```

```
symptom(Patient,conjunctivitis) :-
   write("Does ",Patient," have a conjunctivitis (y/n) ?"),
   response(Reply),
   Reply='y'.

symptom(Patient,cough) :-
   write("Does ",Patient," have a cough (y/n) ?"),
   response(Reply),
   Reply='y'.

symptom(Patient,body_ache) :-
   write("Does ",Patient," have a body_ache (y/n) ?"),
   response(Reply),
   Reply='y'.

symptom(Patient,chills) :-
   write("Does ",Patient," have a chills (y/n) ?"),
   response(Reply),
   Reply='y'.

symptom(Patient,sore_throat) :-
   write("Does ",Patient," have a sore_throat (y/n) ?"),
   response(Reply),
   Reply='y'.

symptom(Patient,sneezing) :-
   write("Does ",Patient," have a sneezing (y/n) ?"),
   response(Reply),
   Reply='y'.

symptom(Patient,swollen_glands) :-
   write("Does ",Patient," have a swollen_glands (y/n) ?"),
   response(Reply),
   Reply='y'.

hypothesis(Patient,measles) :-
   symptom(Patient,fever),
   symptom(Patient,cough),
   symptom(Patient,conjunctivitis),
   symptom(Patient,runny_nose),
   symptom(Patient,rash).

hypothesis(Patient,german_measles) :-
   symptom(Patient,fever),
   symptom(Patient,headache),
```

```
    symptom(Patient,runny_nose),
    symptom(Patient,rash).

hypothesis(Patient,flu) :-
    symptom(Patient,fever),
    symptom(Patient,headache),
    symptom(Patient,body_ache),
    symptom(Patient,conjunctivitis),
    symptom(Patient,chills),
    symptom(Patient,sore_throat),
    symptom(Patient,runny_nose),
    symptom(Patient,cough).

hypothesis(Patient,common_cold) :-
    symptom(Patient,headache),
    symptom(Patient,sneezing),
    symptom(Patient,sore_throat),
    symptom(Patient,runny_nose),
    symptom(Patient,chills).

hypothesis(Patient,mumps) :-
    symptom(Patient,fever),
    symptom(Patient,swollen_glands).

hypothesis(Patient,chicken_pox) :-
    symptom(Patient,fever),
    symptom(Patient,chills),
    symptom(Patient,body_ache),
    symptom(Patient,rash).

hypothesis(Patient,measles) :-
    symptom(Patient,cough),
    symptom(Patient,sneezing),
    symptom(Patient,runny_nose).

response(Reply) :-
    readchar(Reply),
    write(Reply),nl.
```

Output :


PROLOG~1.EXE
Files Edit Run Compile Options Setup
Editor
Line 75 Col 1 C:\USERS\NIRAV\DESKTOP\BORLAN
hypothesis(Patient,measles) :-
symptom(Patient,fever),
symptom(Patient,cough),
symptom(Patient,conjunctivitis),
symptom(Patient,runny_nose),
symptom(Patient,rash).
hypothesis(Patient,german_measles) :-
symptom(Patient,fever),
symptom(Patient,headache),
symptom(Patient,runny_nose),
symptom(Patient,rash).
Dialog
No
Goal: hypothesis(patient,measles)
Does patient have a fever (y/n) ?y
Does patient have a cough (y/n) ?y
Does patient have a conjunctivitis (y/n) ?y
Does patient have a runny_nose (y/n) ?n
Does patient have a cough (y/n) ?n
No
Goal:
Message
N~1.0\MEDICAL2.PRO
hypothesis
symptom
response
Trace
F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End

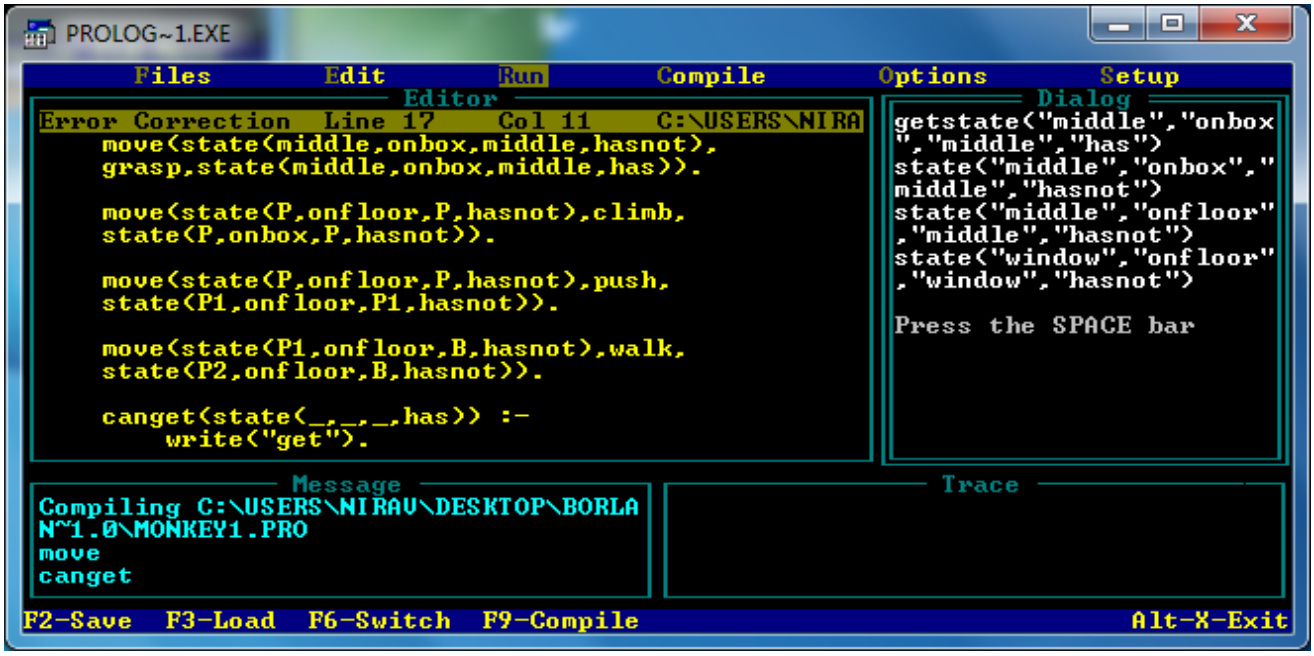
PRACTICAL-8

Aim:- Write a PROLOG/C program to solve Monkey-Banana problem.

Program:-

```
domains
   state=state(symbol,symbol,symbol,symbol)
   /*state=state(monkey horizontal        monkey vertical,        box location,        has/has not   banana)   */

predicates
   move(state,symbol,state)
   canget(state)

clauses
   move(state(middle,onbox,middle,hasnot),
   grasp,state(middle,onbox,middle,has)).

   move(state(P,onfloor,P,hasnot),climb,
   state(P,onbox,P,hasnot)).

   move(state(P,onfloor,P,hasnot),push,
   state(P1,onfloor,P1,hasnot)).

   move(state(P1,onfloor,B,hasnot),walk,
   state(P2,onfloor,B,hasnot)).

   canget(state(_,_,_,has)) :-
      write("get").

   canget(State1) :-
      move(State1,Move,State2),
      canget(State2),
      write(State2),nl.

goal
   clearwindow,
   canget(state(door,onfloor,window,hasnot)).
```

Output:-


PROLOG~1.EXE
Files Edit Run Compile Options Setup
Editor
Error Correction Line 17 Col 11 C:\USERS\NIRA
move(state(middle,onbox,middle,hasnot),
grasp,state(middle,onbox,middle,has)).
move(state(P,onfloor,P,hasnot),climb,
state(P,onbox,P,hasnot)).
move(state(P,onfloor,P,hasnot),push,
state(P1,onfloor,P1,hasnot)).
move(state(P1,onfloor,B,hasnot),walk,
state(P2,onfloor,B,hasnot)).
canget(state(_,_,_,has)) :-
write("get").
Dialog
getstate("middle","onbox
","middle","has")
state("middle","onbox","
middle","hasnot")
state("middle","onfloor"
,"middle","hasnot")
state("window","onfloor"
,"window","hasnot")
Press the SPACE bar
Message
Compiling C:\USERS\NIRAV\DESKTOP\BORLA
N~1.0\MONKEY1.PRO
move
canget
Trace
F2-Save F3-Load F6-Switch F9-Compile Alt-X-Exit

PRACTICAL - 9

Aim:- Write a PROLOG/C program to solve the Travelling Salesman Problem.

Program:-

```
/* About this algorithm:
* Here we use dynamic program
ming to find a solution to&nbs
p;the

* travelling salesperson problem.
The problem consists of finding

* the least-cost cycle in a&n
bsp;given set of nodes. */

#include <stdio.h>
#include <conio.h>
#define MAX 100
#define INFINITY 999

int tsp_dp (int c[][MAX], int tour[], int start, int n);

int main()
{
        int n; /* Number of cities.*/
        int i, j; /* Loop counters.*/
        int c[MAX][MAX]; /* Cost matrix.*/
        int tour[MAX]; /* Tour matrix.*/
        int cost; /* Least cost. */
        clrscr();
        printf ("This program demonstrates the TSP problem.");
        printf ("\nHow many cities to traverse? ");
        scanf ("%d", &n);
        printf ("Enter the cost matrix:(999: no connection)\n");

        for (i=0; i<n; i++)
                for (j=0; j<n; j++)
                        scanf ("%d", &c[i][j]);

        for (i=0; i<n; i++)
                tour[i] = i;

        cost = tsp_dp (c, tour, 0,n);
```

```
        printf ("Minimum cost: %d.\nTour:", cost);

        for (i=0; i<n; i++)
                printf ("%d ", tour[i]+1);
        printf ("1\n");
        getch();
}

int tsp_dp (int c[][MAX], int tour[], int start, int n)
{
        int i, j, k; /* Loop counters. */
        int temp[MAX]; /* Temporary during calculations. */
        int mintour[MAX]; /* Minimal tour array. */
        int mincost; /* Minimal cost.*/
        int ccost; /* Current cost. */

        /* End of recursion condition.*/
        if (start == n - 2)
        {
                printf("\nStart - %d: [%d, %d]%d + [%d, 0]%d = %d\n",start, tour[n-2], tour[n-1], c[tour[n-
                2]][tour[n-1]], tour[n-1], c[tour[n-1]][0], c[tour[n-2]][tour[n-1]] + c[tour[n-1]][0]);
                return c[tour[n-2]][tour[n-1]] + c[tour[n-1]][0];
        }

        /* Compute the tour starting from the current city. */
        mincost = INFINITY;
        for (i = start+1; i<n; i++)
        {
                for (j=0; j<n; j++)
                        temp[j] = tour[j];

                /* Adjust positions. */
                temp[start+1] = tour[i];
                temp[i] = tour[start+1];

                /* Found a better cycle? (Recurrence derivable.) */
                if (c[tour[start]][tour[i]] + (ccost = tsp_dp (c, temp, start+1, n)) < mincost)
                {
                        mincost = c[tour[start]][tour[i]] + ccost;
                        printf("\nmincost: [%d, %d]%d + %d = %d\n",tour[start], tour[i],
        c[tour[start]][tour[i]], ccost, mincost);
                        for (k=0; k<n; k++)
                                mintour[k] = temp[k];
                }
        }
```

```
        /* Set the minimum-tour array.*/
        for (i=0; i<n; i++)
                tour[i] = mintour[i];

        return mincost;
}
```

Output:-

```
This program demonstrates the TSP problem.
How many cities to traverse? 3
Enter the cost matrix:(999: no connection)
1 5 7
2 4 8
3 5 7

Start - 1: [1, 2]8 + [2, 0]3 = 11

mincost: [0, 1]5 + 11 = 16

Start - 1: [2, 1]5 + [1, 0]2 = 7

mincost: [0, 2]7 + 7 = 14
Minimum cost: 14.
Tour:1 3 2 1
```

PRACTICAL-10

Aim:- Write a PROLOG Program to generate query based on family relationship.

Program:-

```
trace
predicates
        person(symbol)
        parent(symbol,symbol)
        sibling(symbol,symbol)
        male(symbol)
        female(symbol)
        brother(symbol,symbol)
        sister(symbol,symbol)
        father(symbol,symbol)
        mother(symbol,symbol)
clauses
        person(a).
        person(b).
        person(c).
        person(d).
        person(e).
        person(u).
        male(a).
        male(e).
        male(u).
        female(b).
        female(c).
        female(d).
        sibling(a,u).
        sibling(c,e).
        sibling(c,d).
        sibling(d,c).
        sibling(d,e).
        sibling(e,c).
        sibling(e,d).
        sibling(u,a).
        parent(a,c).
        parent(a,d).
        parent(a,e).
        parent(b,c).
        parent(b,d).
        parent(b,e).
```

```
brother(A,B):-
        sibling(A,B),male(A),write(A,"brother of",B),nl.
sister(A,B):-
        sibling(A,B),female(A),write(A,"sister of",B),nl.
father(A,B):-
        parent(A,B),male(A),write(A,"father of",B),nl.
mother(A,B):-
        parent(A,B),female(A),write(A,"mother of",B),nl.
```

Output:-