Artificial Intelligence

Practical list

1. Write a c program to implement Tic-Tac-Toe game problem.
2. Write a program to implement BFS (for 8 puzzle problem or Water Jug problem or any AI search problem
3. Write a program to implement DFS (for 8 puzzle problem or Water Jug problem or any AI search problem.
4. Write a program to implement Single Player Game

(Using Heuristic Function)

5. Write a program to Implement A* Algorithm

6. Assume given a set of facts of the form father(name1,name2)

(name1 is the father of name2).

7. Define a predicate brother(X,Y) which holds iff X and Y are brothers.
8. Write a program to solve towers of Hanoi problem

using Prolog.

9. Write a program to solve N-Queens problem using Prolog.
10. Write a program to solve 8 puzzle problem using Prolog.
11. Write a program to solve travelling salesman problem using Prolog.

Practical 1

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

Input:

```
#include<stdio.h>

#include<conio.h>

char square[10] = { 'o', '1', '2', '3', '4', '5', '6', '7', '8', '9' };

int checkwin();

void board();

int main()

{

int player = 1, i, choice;

char mark;

do

{

board();

player = (player % 2) ? 1 : 2;

printf("Player %d, enter a number: ", player);

scanf("%d", &choice);

mark = (player == 1) ? 'X' : 'O';

if (choice == 1 && square[1] == '1')

square[1] = mark;

else if (choice == 2 && square[2] == '2')
```

```
square[2] = mark;
else if (choice == 3 && square[3] == '3')
square[3] = mark;


else if (choice == 4 && square[4] == '4')
square[4] = mark;
else if (choice == 5 && square[5] == '5')
square[5] = mark;
else if (choice == 6 && square[6] == '6')
square[6] = mark;
else if (choice == 7 && square[7] == '7')
square[7] = mark;
else if (choice == 8 && square[8] == '8')
square[8] = mark;
else if (choice == 9 && square[9] == '9')
square[9] = mark;
else
{
printf("Invalid move ");
player--;
getch();
}
```

```
i = checkwin();

player++;

}while (i == - 1);

board();

if (i == 1)

printf("==>\aPlayer %d win ", --player);

else

printf("==>\aGame draw");

getch();

return 0;


}

/*********************************************

FUNCTION TO RETURN GAME STATUS

1 FOR GAME IS OVER WITH RESULT

-1 FOR GAME IS IN PROGRESS

O GAME IS OVER AND NO RESULT

**********************************************/

int checkwin()

{

if (square[1] == square[2] && square[2] == square[3])

return 1;
```

```
else if (square[4] == square[5] && square[5] == square[6])

return 1;

else if (square[7] == square[8] && square[8] == square[9])

return 1;

else if (square[1] == square[4] && square[4] == square[7])

return 1;

else if (square[2] == square[5] && square[5] == square[8])

return 1;

else if (square[3] == square[6] && square[6] == square[9])

return 1;

else if (square[1] == square[5] && square[5] == square[9])

return 1;

else if (square[3] == square[5] && square[5] == square[7])

return 1;

else if (square[1] != '1' && square[2] != '2' && square[3] != '3' &&

square[4] != '4' && square[5] != '5' && square[6] != '6' && square[7]

!= '7' && square[8] != '8' && square[9] != '9')


return 0;

else

return - 1;

}
```

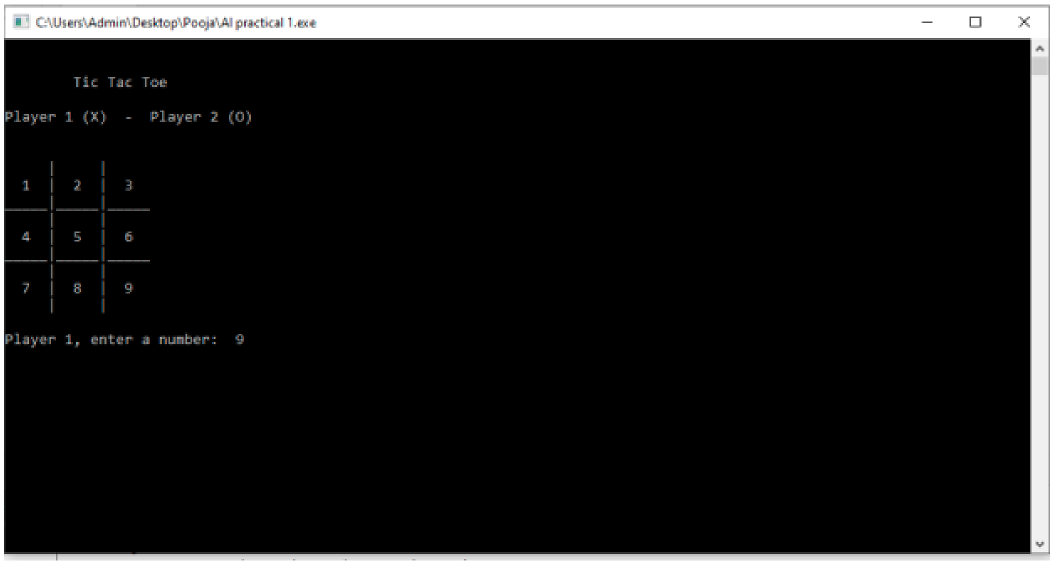
```
/*******************************************************************
*
FUNCTION TO DRAW BOARD OF TIC TAC TOE WITH PLAYERS MARK
*******************************************************************
**/

void board()
{
system("cls");
printf("\n\n\tTic Tac Toe\n\n");
printf("Player 1 (X) - Player 2 (O)\n\n\n");

printf(" | | \n");
printf(" %c | %c | %c \n", square[1], square[2], square[3]);
printf("_____|_____|_____\n");
printf(" | | \n");
printf(" %c | %c | %c \n", square[4], square[5], square[6]);
printf("_____|_____|_____\n");
printf(" | | \n");
printf(" %c | %c | %c \n", square[7], square[8], square[9]);
printf(" | | \n\n");
}
```

/***
*

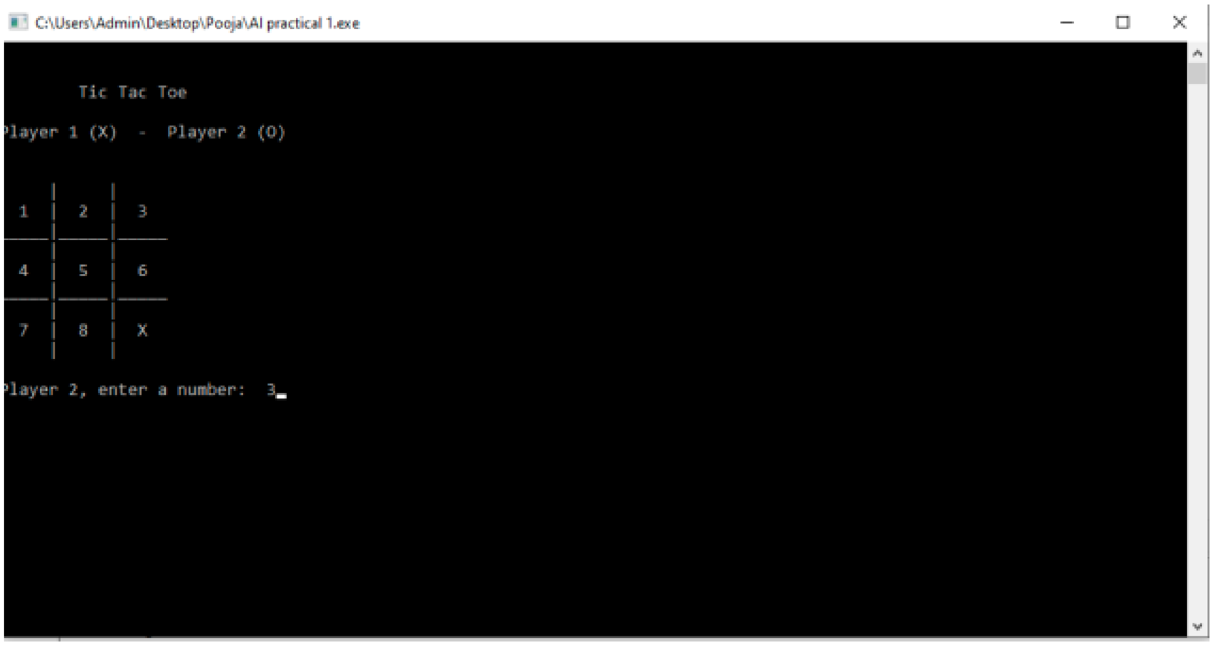
END OF PROJECT

****
**/

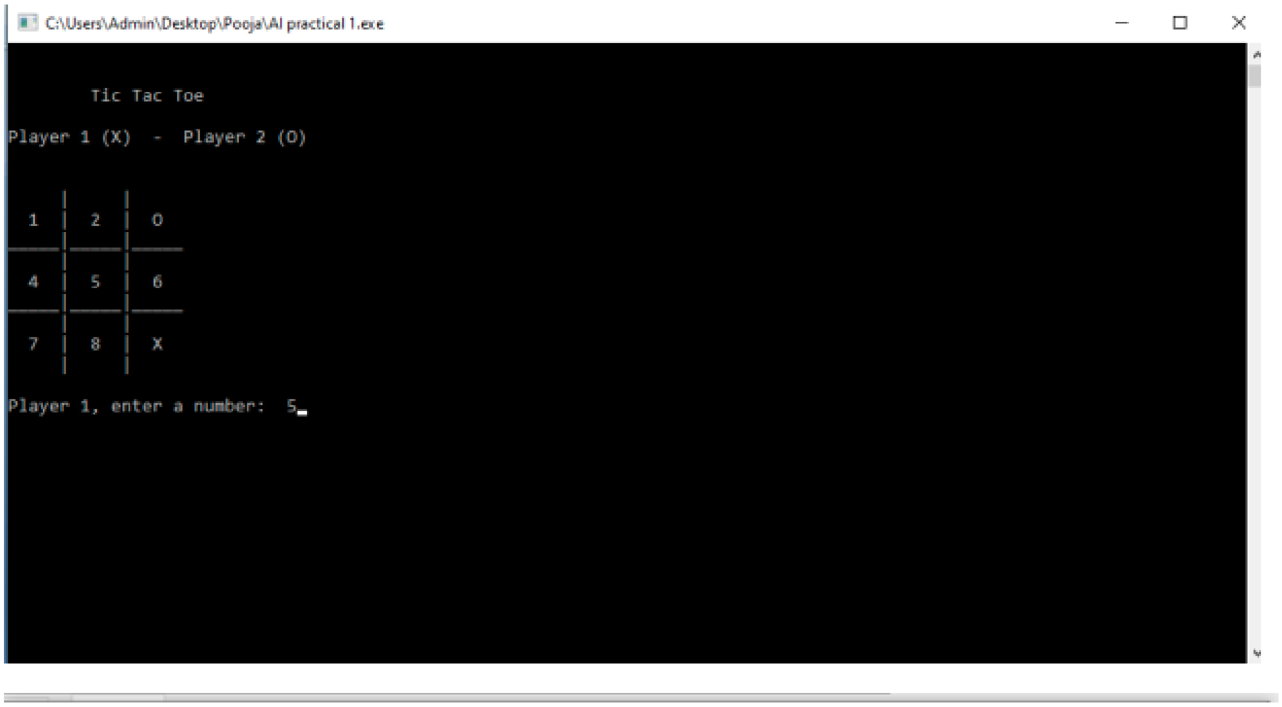
Output:



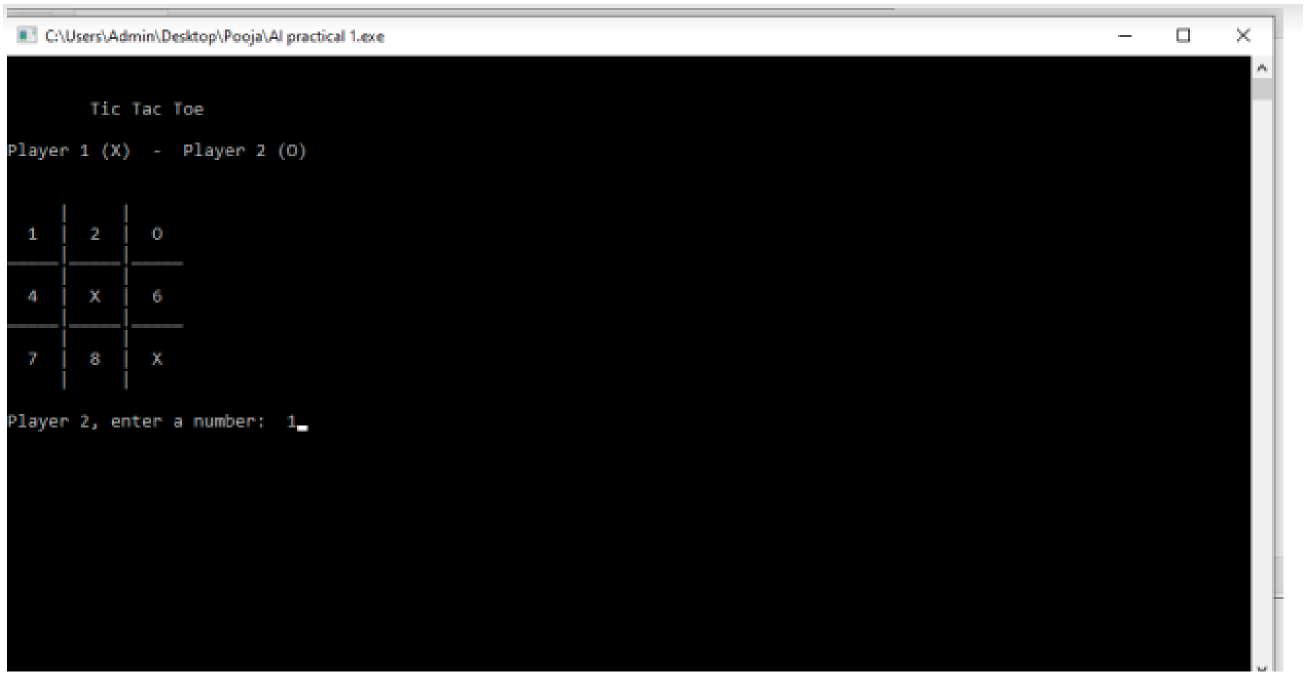
```
C:\Users\Admin\Desktop\Pooja\AI practical 1.exe

        Tic Tac Toe

Player 1 (X)  -  Player 2 (O)


     |     |
  1  |  2  |  3
_____|_____|_____
     |     |
  4  |  5  |  6
_____|_____|_____
     |     |
  7  |  8  |  9
     |     |

Player 1, enter a number:  9
```



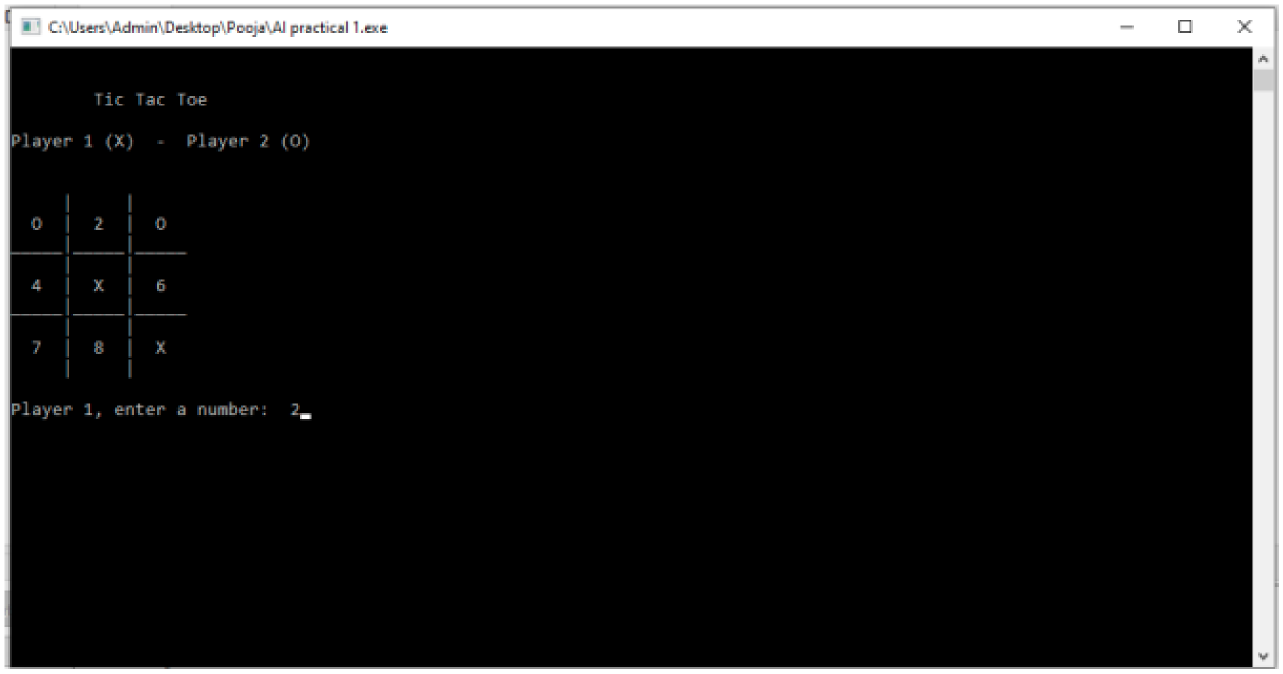
```
C:\Users\Admin\Desktop\Pooja\AI practical 1.exe

        Tic Tac Toe

Player 1 (X)  -  Player 2 (O)


     |     |
  1  |  2  |  3
_____|_____|_____
     |     |
  4  |  5  |  6
_____|_____|_____
     |     |
  7  |  8  |  X
     |     |

Player 2, enter a number:  3
```



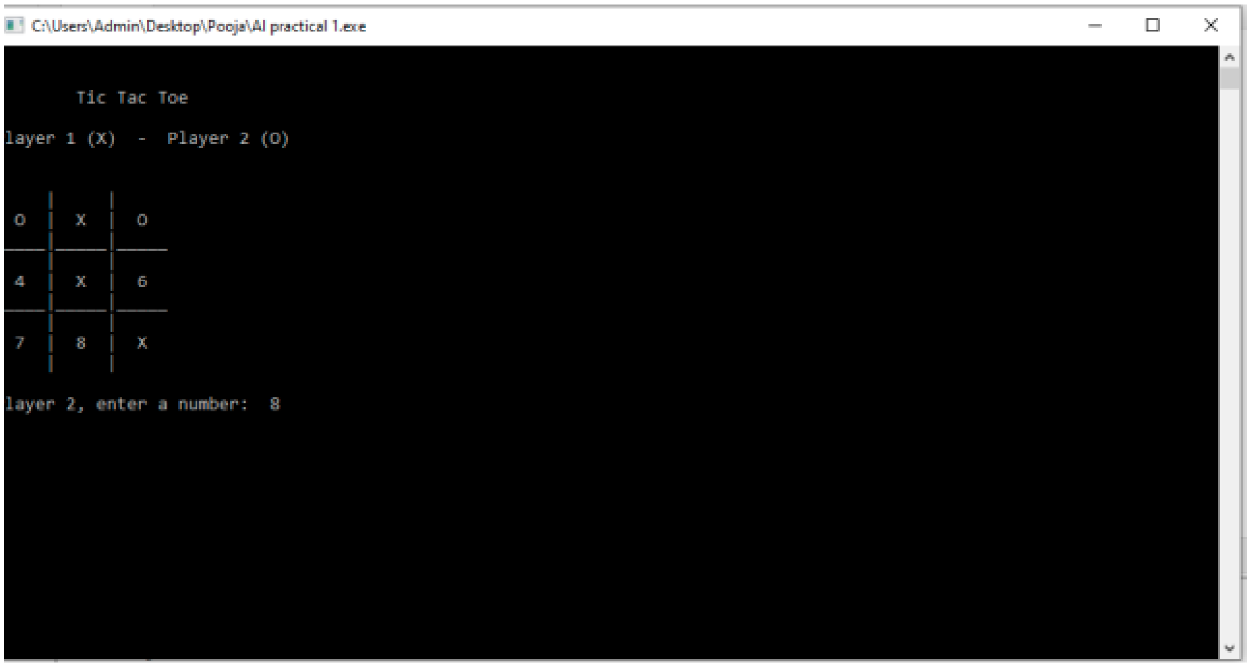
```
C:\Users\Admin\Desktop\Pooja\AI practical 1.exe

        Tic Tac Toe

Player 1 (X)  -  Player 2 (O)


     |     |
  1  |  2  |  O
_____|_____|_____
     |     |
  4  |  5  |  6
_____|_____|_____
     |     |
  7  |  8  |  X
     |     |

Player 1, enter a number:  5
```



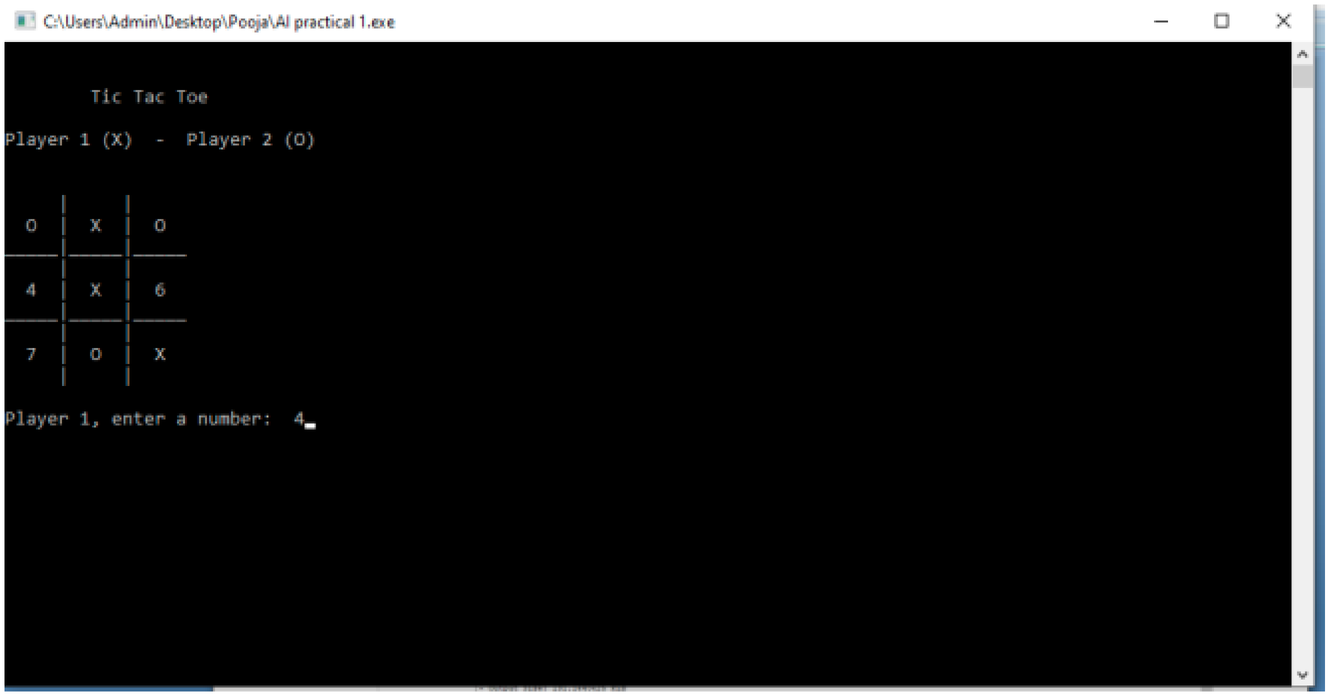
```
C:\Users\Admin\Desktop\Pooja\AI practical 1.exe

        Tic Tac Toe

Player 1 (X)  -  Player 2 (O)


     |     |
  1  |  2  |  O
_____|_____|_____
     |     |
  4  |  X  |  6
_____|_____|_____
     |     |
  7  |  8  |  X
     |     |

Player 2, enter a number:  1
```



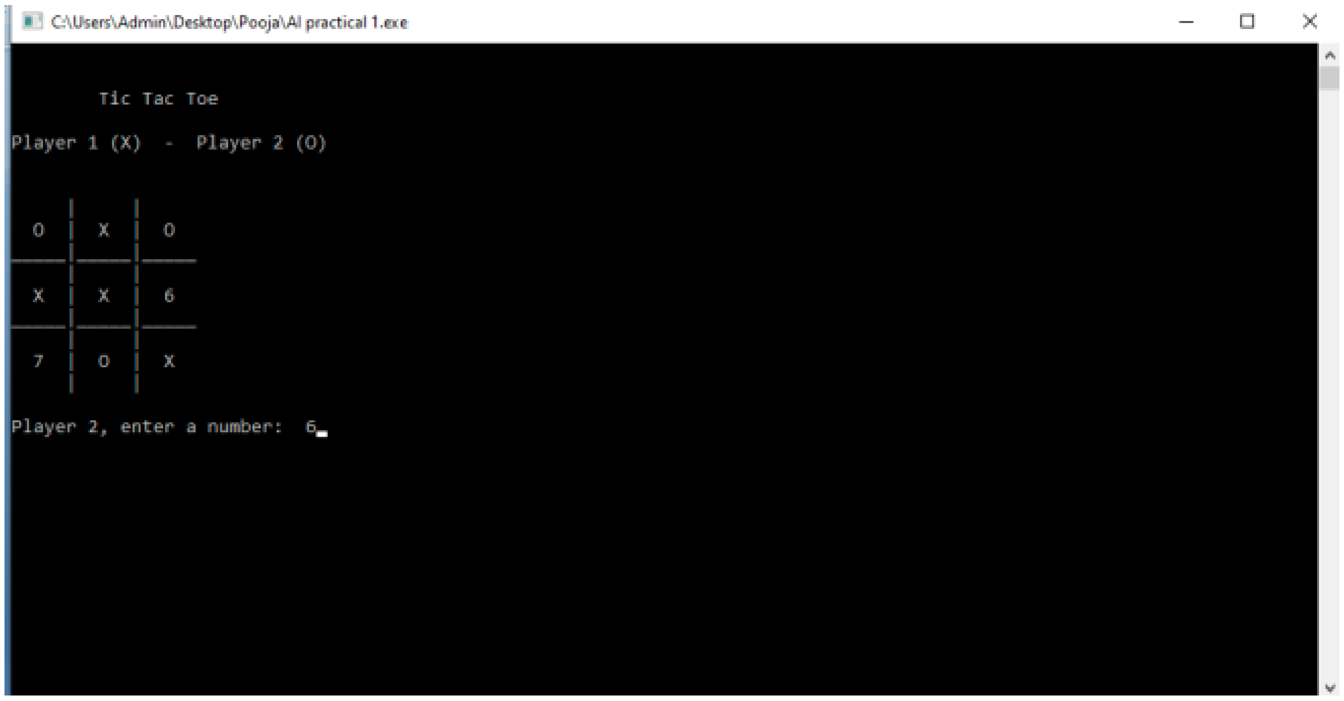
```
C:\Users\Admin\Desktop\Pooja\AI practical 1.exe

        Tic Tac Toe

Player 1 (X)  -  Player 2 (O)


     |     |
  O  |  2  |  O
_____|_____|_____
     |     |
  4  |  X  |  6
_____|_____|_____
     |     |
  7  |  8  |  X
     |     |

Player 1, enter a number:  2
```


C:\Users\Admin\Desktop\Pooja\AI practical 1.exe

```
      Tic Tac Toe

layer 1 (X)  -  Player 2 (O)


     |     |
  O  |  X  |  O
_____|_____|_____
     |     |
  4  |  X  |  6
_____|_____|_____
     |     |
  7  |  8  |  X
     |     |

layer 2, enter a number:  8
```


C:\Users\Admin\Desktop\Pooja\AI practical 1.exe

```
        Tic Tac Toe

Player 1 (X)  -  Player 2 (O)


     |     |
  O  |  X  |  O
_____|_____|_____
     |     |
  4  |  X  |  6
_____|_____|_____
     |     |
  7  |  O  |  X
     |     |

Player 1, enter a number:  4
```


C:\Users\Admin\Desktop\Pooja\AI practical 1.exe

```
        Tic Tac Toe

Player 1 (X)  -  Player 2 (O)


     |     |
  O  |  X  |  O
_____|_____|_____
     |     |
  X  |  X  |  6
_____|_____|_____
     |     |
  7  |  O  |  X
     |     |

Player 2, enter a number:  6_
```

Practical 2

- **AIM**: Write a program to implement BFS (for 8 puzzle problem or Water Jug problem or any AI search problem.

Program:

```
#include&lt;stdio.h&gt;
#include&lt;conio.h&gt;
struct node
{
int x, y;
struct node *next;
}*root, *left, *right;

int isNodePresent(struct node *next, int jug1, int jug2, int f1, int f2)
{
struct node *temp;
if((next-&gt;x == f1) &amp;&amp; (next-&gt;y == f2)){
return(0);
}
if((next-&gt;x == jug1) &amp;&amp; (next-&gt;y == jug2)){
return(1);
}
if((next-&gt;x == 0) &amp;&amp; (next-&gt;y == 0)){
return(1);
}
temp = left;
while(1)
{
if((temp-&gt;x == next-&gt;x) &amp;&amp; (temp-&gt;y == next-&gt;y)){
return(1);
}
else if(temp-&gt;next == NULL){
break;
}
else{
temp = temp-&gt;next;
}
```

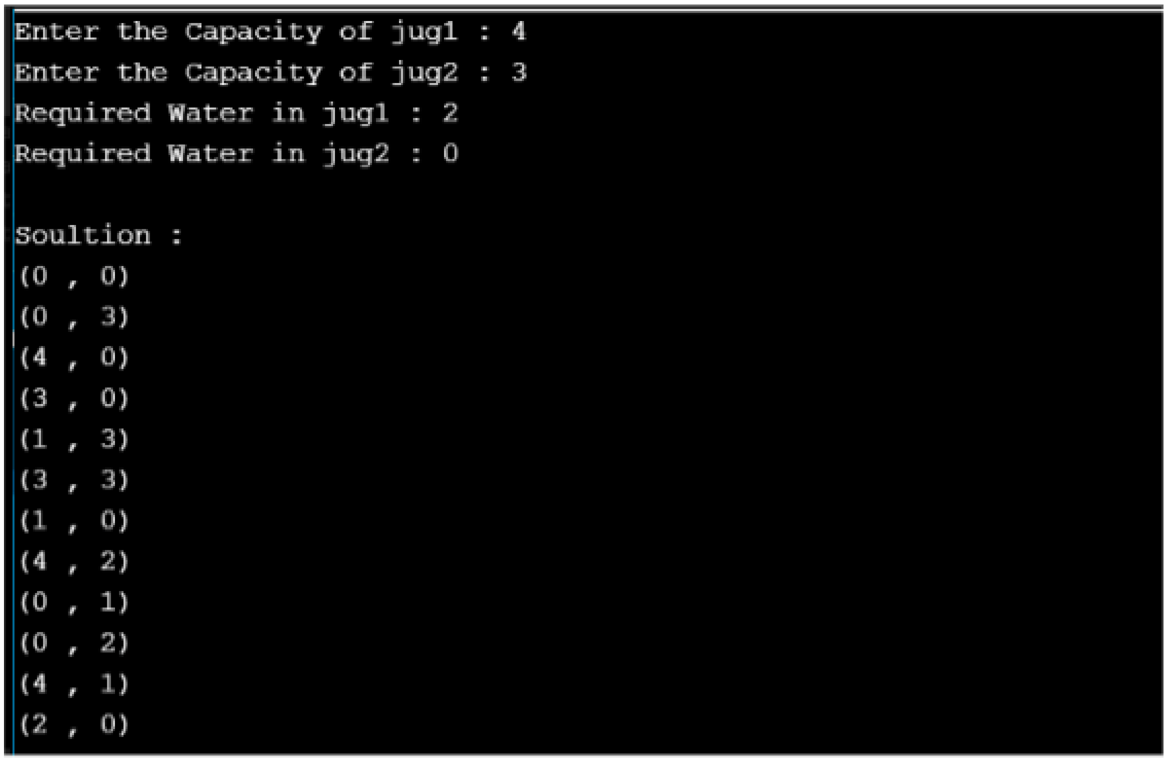
```
}
temp = right;
while(1)
{
if((temp-&gt;x == next-&gt;x) &amp;&amp; (temp-&gt;y == next-&gt;y)){
return(1);
}
else if(temp-&gt;next == NULL){
break;
}
temp = temp-&gt;next;
}
return(0);
}
void bfstree(int jug1, int jug2, int f1, int f2)
{
int flag1, flag2;
struct node *tempLeft, *tempRight;
root = (struct node*)malloc(sizeof(struct node));
root-&gt;x = 0; root-&gt;y = 0; root-&gt;next = NULL;
left = (struct node*)malloc(sizeof(struct node));
left-&gt;x = 0; left-&gt;y = jug2; left-&gt;next = NULL;
right = (struct node*)malloc(sizeof(struct node));
right-&gt;x = jug1; right-&gt;y = 0; right-&gt;next = NULL;
tempLeft = left;
tempRight = right;
while(1)
{
flag1 = 0; flag2 = 0;
if((tempLeft-&gt;x != f1) || (tempLeft-&gt;y != f2)) {
tempLeft-&gt;next = genNewState(tempLeft, jug1, jug2, f1, f2);
tempLeft = tempLeft-&gt;next;
tempLeft-&gt;next = NULL;
flag1 = 1;
}
if((tempRight-&gt;x != f1) || (tempRight-&gt;y != f2)) {
tempRight-&gt;next = genNewState(tempRight, jug1, jug2, f1, f2);
tempRight = tempRight-&gt;next;
tempRight-&gt;next = NULL;
```

```
flag2 = 1;
}
if((flag1 == 0) &amp;&amp; (flag2 == 0)){
break;
}
}
}
struct node* genNewState(struct node *current, int jug1, int jug2, int f1, int f2)
{
int d;
struct node *next;
next = (struct node*)malloc(sizeof(struct node));
next-&gt;x = jug1;
next-&gt;y = current-&gt;y;
if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
return(next);
}
next-&gt;x = current-&gt;x;
next-&gt;y = jug2;
if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
return(next);
}
next-&gt;x = 0;
next-&gt;y = current-&gt;y;
if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
return(next);
}
next-&gt;y = 0;
next-&gt;x = current-&gt;x;
if(isNodePresent(next, jug1, jug2, f1, f2) != 1) {
return(next);
}
if((current-&gt;y &lt; jug2) &amp;&amp; (current-&gt;x != 0)) {
d = jug2 - current-&gt;y;
if(d &gt;= current-&gt;x) {
next-&gt;x = 0;


next-&gt;y = current-&gt;y + current-&gt;x;
```

```
} else {
next-&gt;x = current-&gt;x - d;
next-&gt;y = current-&gt;y + d;
}
if(isNodePresent(next, jug1, jug2, f1, f2) != 1) {
return(next);
}
}
if((current-&gt;x &lt; jug1) &amp;&amp; (current-&gt;y != 0)) {
d = jug1 - current-&gt;x;
if(d &gt;= current-&gt;y) {
next-&gt;y = 0;
next-&gt;x = current-&gt;x + current-&gt;y;
}
else
{
next-&gt;y = current-&gt;y - d;
next-&gt;x = current-&gt;x + d;
}
if(isNodePresent(next, jug1, jug2, f1, f2) != 1) {
return(next);
}
}
return(NULL);
}
void BFS(int f1, int f2)
{
struct node *temp1 = left, *temp2 = right;
printf(&quot;\nSoultion : \n&quot;);
printf(&quot;(%d , %d)\n&quot;, root-&gt;x, root-&gt;y);
while(1)
{
printf(&quot;(%d , %d)\n&quot;, temp1-&gt;x, temp1-&gt;y);
if((temp1-&gt;x == f1)&amp;&amp;(temp1-&gt;y == f2)){
break;
}


temp1 = temp1-&gt;next;
```

```
printf(&quot;(%d , %d)\n&quot;, temp2-&gt;x, temp2-&gt;y);
if((temp2-&gt;x == f1)&amp;&amp;(temp2-&gt;y == f2)){
break;
}
temp2 = temp2-&gt;next;
}
}
void main()
{
int jug1, jug2, f1, f2;
clrscr();
printf(&quot;Enter the Capacity of jug1 : &quot;);
scanf(&quot;%d&quot;, &amp;jug1);
printf(&quot;Enter the Capacity of jug2 : &quot;);
scanf(&quot;%d&quot;, &amp;jug2);
printf(&quot;\nRequired Water in jug1 : &quot;);
scanf(&quot;%d&quot;, &amp;f1);
printf(&quot;Required Water in jug2 : &quot;);
scanf(&quot;%d&quot;, &amp;f2);
bfstree(jug1, jug2, f1, f2);
BFS(f1, f2);
getch();
}
```

Output:



```
Enter the Capacity of jug1 : 4
Enter the Capacity of jug2 : 3
Required Water in jug1 : 2
Required Water in jug2 : 0

Soultion :
(0 , 0)
(0 , 3)
(4 , 0)
(3 , 0)
(1 , 3)
(3 , 3)
(1 , 0)
(4 , 2)
(0 , 1)
(0 , 2)
(4 , 1)
(2 , 0)
```

Practical 3

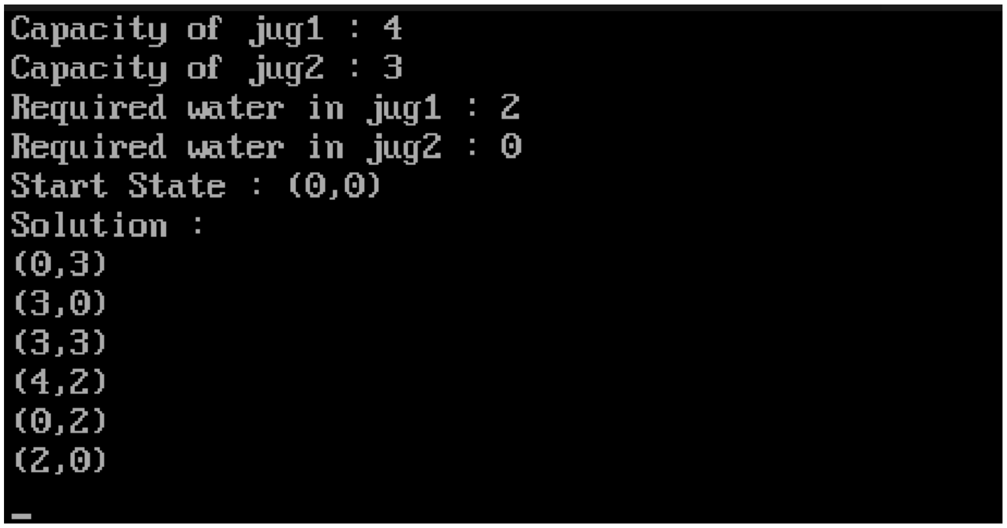
- **AIM:** Write a program to implement DFS (for 8 puzzle problem or Water Jug problem or any AI search problem.

  Program:

```
#include<stdio.h>
#include<conio.h>
struct node
{
   int x, y;
   struct node *next;
}*root, *left, *right;

void main()
{
   int jug1, jug2, f1, f2;
   clrscr();
   printf("Capacity of jug1 : ");
   scanf("%d", &jug1);
   printf("Capacity of jug2 : ");
   scanf("%d", &jug2);
   printf("Required water in jug1 : ");
   scanf("%d", &f1);
   printf("Required water in jug2 : ");
   scanf("%d", &f2);
   generateTree(jug1, jug2, f1, f2);
   DFS();
   getch();
}

int isNodePresent(struct node *next, int jug1, int jug2, int f1, int f2)
{
   struct node *temp;
   if((next->x == f1) && (next->y == f2)){
      return(0);
   }
   if((next->x == jug1) && (next->y == jug2)){
      return(1);
   }
```

```
    if((next->x == 0) && (next->y == 0)){
        return(1);
    }
    temp = left;
    while(1)
    {
        if((temp->x == next->x) && (temp->y == next->y)){
            return(1);
        }
        else if(temp->next == NULL){
            break;
        } else {
            temp = temp->next;
        }
    }
    temp = right;
    while(1)
    {
        if((temp->x == next->x) && (temp->y == next->y)){
            return(1);
        } else if(temp->next == NULL){
            break;
        }
        temp = temp->next;
    }
    return(0);
}

void DFS()
{
    struct node *temp;
    temp = left;
    printf("Start State : (%d,%d)\n", root->x, root->y);
    printf("Solution : \n");
    while(1)
    {
        printf("(%d,%d)\n", temp->x, temp->y);
        if(temp->next == NULL){
            break;
        }
        temp = temp->next;
    }
    temp = right;
}


struct node* genNewState(struct node *current, int jug1, int jug2, int f1, int f2)
{
    int d;
    struct node *next;
    next = (struct node*)malloc(sizeof(struct node));
    next->x = jug1;
    next->y = current->y;
    if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
        return(next);
    }
    next->x = current->x;
    next->y = jug2;
    if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
        return(next);
    }
    next->x = 0;
    next->y = current->y;
    if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
        return(next);
    }
    next->y = 0;
    next->x = current->x;
    if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
        return(next);
    }
    if((current->y < jug2) && (current->x != 0))
    {
        d = jug2 - current->y;
        if(d >= current->x)
        {
            next->x = 0;
            next->y = current->y + current->x;
        } else {
            next->x = current->x - d;
            next->y = current->y + d;
        }
        if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
            return(next);
        }
    }
    if((current->x < jug1) && (current->y != 0))
    {
        d = jug1 - current->x;
        if(d >= current->y) {
            next->y = 0;
            next->x = current->x + current->y;
        } else {
            next->y = current->y - d;
            next->x = current->x + d;
        }
        if(isNodePresent(next, jug1, jug2, f1, f2) != 1){
            return(next);
        }
    }
    return(NULL);
}

void generateTree(int jug1, int jug2, int f1, int f2)
{
    int flag1, flag2;
    struct node *tempLeft, *tempRight;
    root  = (struct node*)malloc(sizeof(struct node));
    root->x = 0; root->y = 0; root->next = NULL;
    left = (struct node*)malloc(sizeof(struct node));
    left->x = 0; left->y = jug2; left->next = NULL;
    right = (struct node*)malloc(sizeof(struct node));
    right->x = jug1; right->y = 0; right->next = NULL;
    tempLeft = left;
    tempRight = right;
    while(1)
    {
        flag1 = 0; flag2 = 0;
        if((tempLeft->x != f1) || (tempLeft->y != f2))
        {
           tempLeft->next = genNewState(tempLeft, jug1, jug2, f1, f2);
           tempLeft = tempLeft->next;
           tempLeft->next = NULL;
           flag1 = 1;
        }
        if((tempRight->x != f1) || (tempRight->y != f2))
        {
           tempRight->next = genNewState(tempRight, jug1, jug2, f1, f2);
           tempRight = tempRight->next;
           tempRight->next = NULL;
           flag2 = 1;
        }
        if((flag1 == 0) && (flag2 == 0))
           break;
    }
          }
```

Output:



```
Capacity of jug1 : 4
Capacity of jug2 : 3
Required water in jug1 : 2
Required water in jug2 : 0
Start State : (0,0)
Solution :
(0,3)
(3,0)
(3,3)
(4,2)
(0,2)
(2,0)

_
```

Practical 4

❖ **Aim:** Write a program to implement Single Player Game (Using Heuristic Function)

Program:

```
#include <stdio.h>

#include <stdlib.h>

char matrix[3][3]; char check(void);

void init_matrix(void);

void get_player_move(void);

void get_computer_move(void);


void disp_matrix(void);

int main(void)

{

char done;

printf("This is the game of Tic Tac Toe.\n");

printf("You will be playing against the computer.\n");

done = ' ';

init_matrix();

do {

disp_matrix();

get_player_move();
done = check(); /* see if winner */

if(done!= ' ') break; /* winner!*/

get_computer_move();

done = check(); /* see if winner */

} while(done== ' ');

if(done=='X') printf("You won!\n");

else printf("I won!!!!\n");

disp_matrix(); /* show final positions */

return 0;

}

/* Initialize the matrix. */

void init_matrix(void)

{

int i, j;


for(i=0; i<3; i++)

for(j=0; j<3; j++) matrix[i][j] = ' ';

}

/* Get a player's move. */

void get_player_move(void)

{
```

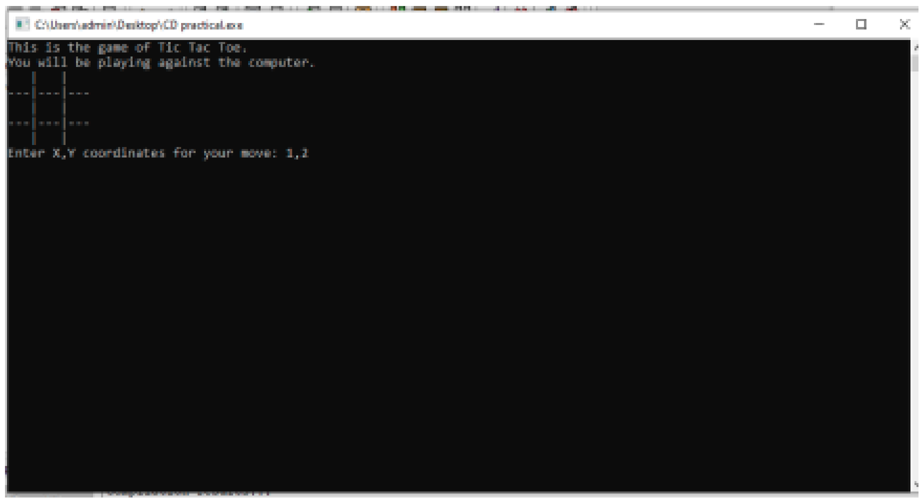
```
int x, y;


printf("Enter X,Y coordinates for your move: "); scanf("%d%*c%d",

&x, &y); x--; y--;


if(matrix[x][y]!= ' '){


printf("Invalid move, try again.\n");

get_player_move();

}

else matrix[x][y] = 'X';

}

/* Get a move from the computer. */

void get_computer_move(void)

{

int i, j;

for(i=0; i<3; i++){

for(j=0; j<3; j++)

if(matrix[i][j]==' ') break;

if(matrix[i][j]==' ') break;

}
```

```
if(i*j==9) {

printf("draw\n");

exit(0);

}

else

matrix[i][j] = 'O';

}


/* Display the matrix on the screen. */

void disp_matrix(void)

{

int t;

for(t=0; t<3; t++) {

printf(" %c | %c | %c ",matrix[t][0],

matrix[t][1], matrix [t][2]);

if(t!=2) printf("\n---|---|---\n");

}

printf("\n");

}


/* See if there is a winner. */
```

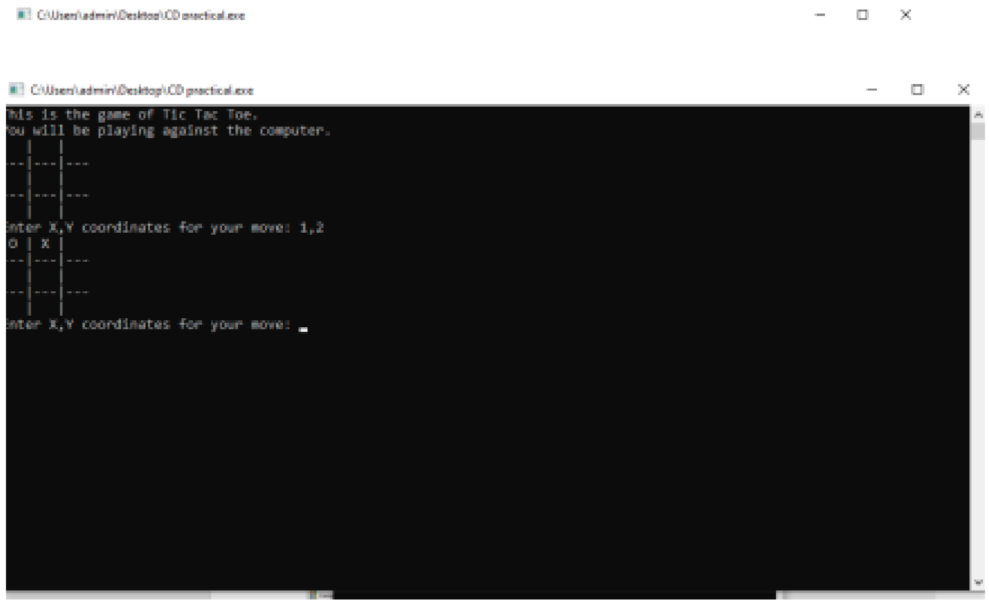
Artificial intelligence Enrollment no.:181010107008

```
char check(void)

{

int i;


for(i=0; i<3; i++) /* check rows */

if(matrix[i][0]==matrix[i][1] &&

matrix[i][0]==matrix[i][2]) return matrix[i][0];

for(i=0; i<3; i++) /* check columns */

if(matrix[0][i]==matrix[1][i] &&

matrix[0][i]==matrix[2][i]) return matrix[0][i];

/* test diagonals */

if(matrix[0][0]==matrix[1][1] &&

matrix[1][1]==matrix[2][2])

return matrix[0][0];

if(matrix[0][2]==matrix[1][1] &&


matrix[1][1]==matrix[2][0])

return matrix[0][2];

return ' ';

}
```

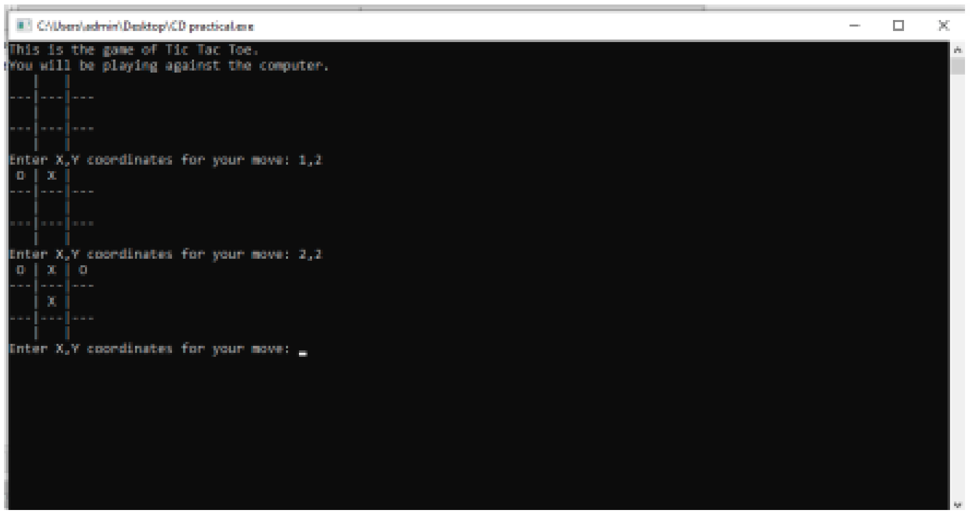
Output:



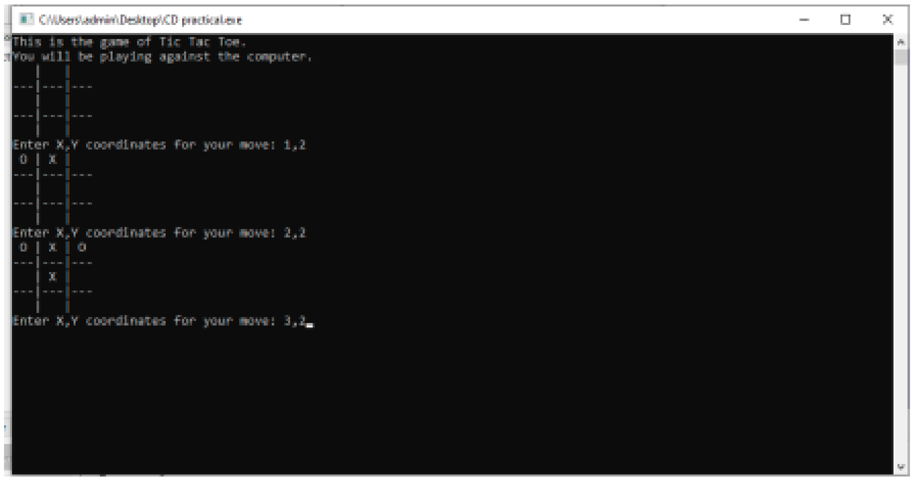
```
This is the game of Tic Tac Toe.
You will be playing against the computer.
   |   |
---|---|---
   |   |
---|---|---
   |   |
Enter X,Y coordinates for your move: 1,2
```


C:\Users\admin\Desktop\CD practical.exe
C:\Users\admin\Desktop\CD practical.exe

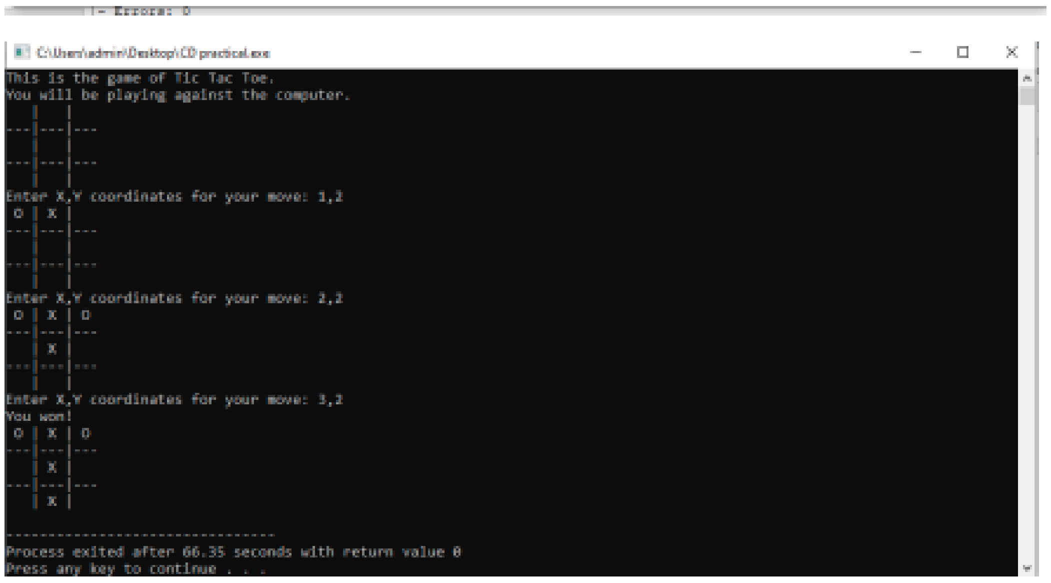
```
his is the game of Tic Tac Toe.
ou will be playing against the computer.
   |   |
---|---|---
   |   |
---|---|---
   |   |
nter X,Y coordinates for your move: 1,2
O | X |
---|---|---
   |   |
---|---|---
   |   |
nter X,Y coordinates for your move: _
```



```
C:\Users\admin\Desktop\CD practical.exe
This is the game of Tic Tac Toe.
You will be playing against the computer.
   |   |
---|---|---
   |   |
---|---|---
   |   |
Enter X,Y coordinates for your move: 1,2
 O | X |
---|---|---
   |   |
---|---|---
   |   |
Enter X,Y coordinates for your move: 2,2
 O | X | O
---|---|---
   | X |
---|---|---
   |   |
Enter X,Y coordinates for your move: _
```



```
C:\Users\admin\Desktop\CD practical.exe
This is the game of Tic Tac Toe.
You will be playing against the computer.
   |   |
---|---|---
   |   |
---|---|---
   |   |
Enter X,Y coordinates for your move: 1,2
 O | X |
---|---|---
   |   |
---|---|---
   |   |
Enter X,Y coordinates for your move: 2,2
 O | X | O
---|---|---
   | X |
---|---|---
   |   |
Enter X,Y coordinates for your move: 3,2_
```



```
|- Errors: 0
```

C:\Users\admin\Desktop\CD practical.exe

```
This is the game of Tic Tac Toe.
You will be playing against the computer.
   |   |
---|---|---
   |   |
---|---|---
   |   |
Enter X,Y coordinates for your move: 1,2
 O | X |
---|---|---
   |   |
---|---|---
   |   |
Enter X,Y coordinates for your move: 2,2
 O | X | O
---|---|---
   | X |
---|---|---
   |   |
Enter X,Y coordinates for your move: 3,2
You won!
 O | X | O
---|---|---
   | X |
---|---|---
   | X |

--------------------------------
Process exited after 66.35 seconds with return value 0
Press any key to continue . . .
```

Practical 5

Aim: Write a program to Implement A* Algorithm

Program:

```
#include <stdlib.h>

#include <stdio.h>


#include <string.h>

#include <float.h>

/* and not not_eq */

#include <iso646.h>

/* add -lm to command line to compile with this header */
#include <math.h>


#define map_size_rows 10

#define map_size_cols 10


char map[map_size_rows][map_size_cols] = {

 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

 {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},

 {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},

 {1, 0, 0, 0, 0, 1, 1, 1, 0, 1},

 {1, 0, 0, 1, 0, 0, 0, 1, 0, 1},
 {1, 0, 0, 1, 0, 0, 0, 1, 0, 1},
 {1, 0, 0, 1, 1, 1, 1, 1, 0, 1},
 {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
 {1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
 };

/* description of graph node */
struct stop {
 double col, row;
 /* array of indexes of routes from this stop to neighbours in array of  all routes
*/
 int * n;
 int n_len;
 double f, g, h;
 int from;
 };

int ind[map_size_rows][map_size_cols] = {
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
 {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1}
 };


/* description of route between two nodes */
struct route {
 /* route has only one direction! */
 int x; /* index of stop in array of all stops of src of this route */  int y; /*
intex of stop in array of all stops od dst of this route */  double d;
 };


int main() {
 int i, j, k, l, b, found;
 int p_len = 0;
 int * path = NULL;
 int c_len = 0;
 int * closed = NULL;
 int o_len = 1;

 int * open = (int*)calloc(o_len, sizeof(int));
 double min, tempg;
 int s;
 int e;
 int current;
 int s_len = 0;
 struct stop * stops = NULL;
 int r_len = 0;
 struct route * routes = NULL;

 for (i = 1; i < map_size_rows - 1; i++) {
 for (j = 1; j < map_size_cols - 1; j++) {
 if (!map[i][j]) {
 ++s_len;
 stops = (struct stop *)realloc(stops, s_len * sizeof(struct stop));  int t = s_len - 1;
 stops[t].col = j;
 stops[t].row = i;
 stops[t].from = -1;
 stops[t].g = DBL_MAX;
 stops[t].n_len = 0;
 stops[t].n = NULL;
 ind[i][j] = t;

 }
 }
 }

 /* index of start stop */
```

```
s = 0;
/* index of finish stop */
e = s_len - 1;


for (i = 0; i < s_len; i++) {
stops[i].h = sqrt(pow(stops[e].row - stops[i].row, 2)
+  pow(stops[e].col - stops[i].col, 2));
}


for (i = 1; i < map_size_rows - 1; i++) {
for (j = 1; j < map_size_cols - 1; j++) {
if (ind[i][j] >= 0) {
for (k = i - 1; k <= i + 1; k++) {
for (l = j - 1; l <= j + 1; l++) {
if ((k == i) and (l == j)) {
continue;
}
if (ind[k][l] >= 0) {

++r_len;
routes = (struct route *)realloc(routes, r_len *  sizeof(struct route));
int t = r_len - 1;
routes[t].x = ind[i][j];
routes[t].y = ind[k][l];
routes[t].d = sqrt(pow(stops[routes[t].y].row - stops[routes[t].x].row, 2)
+ pow(stops[routes[t].y].col - stops[routes[t].x].col, 2));
++stops[routes[t].x].n_len;
stops[routes[t].x].n = (int*)realloc(stops[routes[t].x].n,  stops[routes[t].x].n_len *
sizeof(int));
stops[routes[t].x].n[stops[routes[t].x].n_len - 1] = t;  }
}
}
}
 }
}

open[0] = s;
stops[s].g = 0;
stops[s].f = stops[s].g + stops[s].h;
found = 0;


while (o_len and not found) {
min = DBL_MAX;

for (i = 0; i < o_len; i++) {
if (stops[open[i]].f < min) {
current = open[i];
min = stops[open[i]].f;
}
}
```

```
 if (current == e) {
 found = 1;


 ++p_len;
 path = (int*)realloc(path, p_len * sizeof(int));
 path[p_len - 1] = current;
 while (stops[current].from >= 0) {
 current = stops[current].from;
 ++p_len;
 path = (int*)realloc(path, p_len * sizeof(int));  path[p_len - 1]
= current;
 }
}



 for (i = 0; i < o_len; i++) {
 if (open[i] == current) {
 if (i not_eq (o_len - 1)) {
 for (j = i; j < (o_len - 1); j++) {
 open[j] = open[j + 1];
 }
 }
 --o_len;
 open = (int*)realloc(open, o_len * sizeof(int));  break;
 }
 }
```

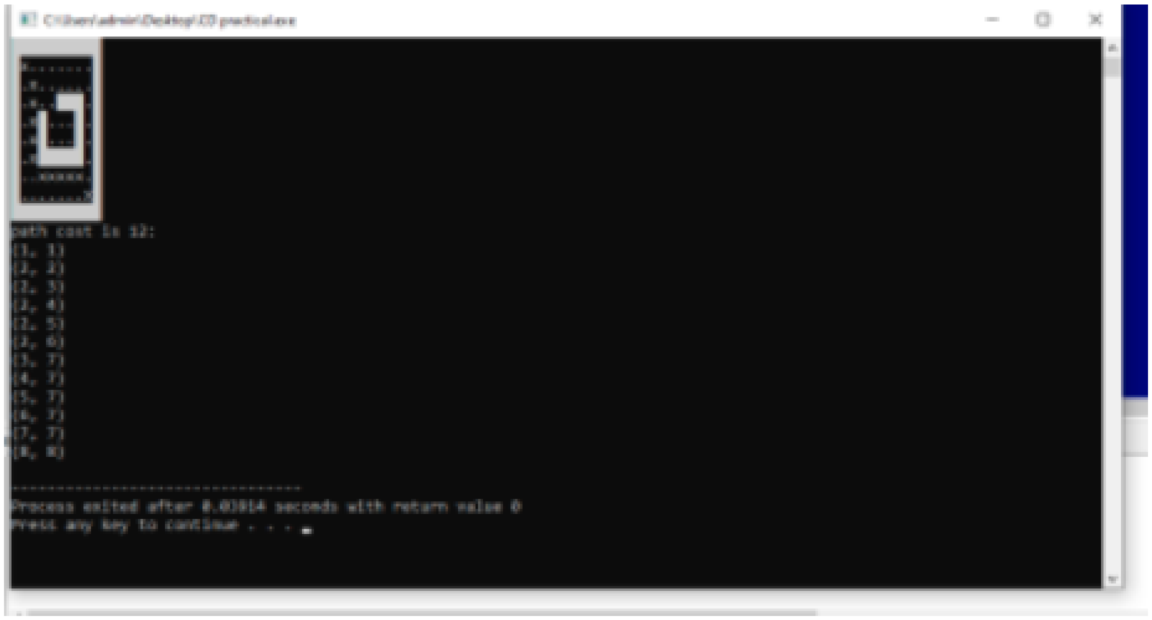
```
++c_len;

closed = (int*)realloc(closed, c_len * sizeof(int));

closed[c_len - 1] = current;


for (i = 0; i < stops[current].n_len; i++) {

b = 0;


for (j = 0; j < c_len; j++) {
```

Artificial Intelligence Enrollment no.:181010107008

```
if (routes[stops[current].n[i]].y == closed[j]) {  b = 1;
}

}


if (b) {

continue;

}


tempg = stops[current].g + routes[stops[current].n[i]].d;  b = 1;


if (o_len > 0) {

for (j = 0; j < o_len; j++) {

if (routes[stops[current].n[i]].y == open[j]) {  b = 0;
}

}

}
```

```
 if (b or (tempg < stops[routes[stops[current].n[i]].y].g))
{  stops[routes[stops[current].n[i]].y].from = current;


 stops[routes[stops[current].n[i]].y].g = tempg;

 stops[routes[stops[current].n[i]].y].f =
stops[routes[stops[current].n[i]].y].g +
stops[routes[stops[current].n[i]].y].h;


 if (b) {

 ++o_len;

 open = (int*)realloc(open, o_len * sizeof(int));  open[o_len - 1] =
routes[stops[current].n[i]].y;  }
 }

 }

 }


 for (i = 0; i < map_size_rows; i++) {

 for (j = 0; j < map_size_cols; j++) {

 if (map[i][j]) {

 putchar(0xdb);

 } else {

 b = 0;

 for (k = 0; k < p_len; k++) {

 if (ind[i][j] == path[k]) {

 ++b;

 }
 }
 if (b) {
 putchar('x');
 } else {
 putchar('.');
 }
 }
 }
 putchar('\n');
 }


 if (not found) {
 puts("IMPOSSIBLE");
 } else {
 printf("path cost is %d:\n", p_len);
 for (i = p_len - 1; i >= 0; i--) {
 printf("(%1.0f, %1.0f)\n", stops[path[i]].col, stops[path[i]].row);  }
 }


 for (i = 0; i < s_len; ++i) {
 free(stops[i].n);
 }


 free(stops);
 free(routes);
 free(path);
 free(open);
 free(closed);


 return 0;
 }
```

Output:



```
[illegible]
path cost is 12:
(1, 1)
(2, 2)
(2, 3)
(2, 4)
(2, 5)
(2, 6)
(3, 7)
(4, 7)
(5, 7)
(6, 7)
(7, 7)
(8, 8)

--------------------------------
Process exited after 0.03014 seconds with return value 0
Press any key to continue . . .
```

Practical 6

❖ **Aim**: Assume given a set of facts of the form father(name1,name2) (name1 is the father of name2).

- Define a predicate sibling(X,Y) which holds if X and Y are siblings.

sibling(X,Y) :- parent(Z,X), parent(Z,Y), not(X=Y).

- Define a predicate cousin(X,Y) which holds if X and Y are cousins.

cousin(X,Y) :- parent(Z,X), parent(W,Y), sibling(Z,W).

- Define a predicate grandchild(X,Y) which holds if X is a grandchild of Y.

grandchild(X,Y) :- parent(Z,X), parent(Y,Z).

- Define a predicate descendent(X,Y) which holds if X is a descendent of Y.

descendent(X,Y) :- parent(Y,X).

descendent(X,Y) :- parent(Z,X), descendent(Z,Y).

Practical 7

❖Aim: Define a predicate brother(X,Y) which holds iff X and Y are

brothers.

Define a predicate cousin(X,Y) which holds iff X and Y are

cousins.

Define a predicate grandson(X,Y) which holds iff X is a grandson

of Y.

Define a predicate descendent(X,Y) which holds iff X is a

descendent of Y.

Consider the following genealogical tree:

father(a,b).

father(a,c).

father(b,d). father(b,e).

father(c,f).

Say which answers, and in which order, are generated by your

definitions for the following queries in Prolog: ?-

brother(X,Y). ?-

cousin(X,Y). ?-

grandson(X,Y). ?-

descendent(X,Y).

solution:

AI(3170716) Enrollment no.:181010107008

o Consider the following genealogical tree:

father(a,b). % 1

father(a,c). % 2

father(b,d). % 3

father(b,e). % 4

father(c,f). % 5

❖ whose graphical representation is:

a

**/ **

b c

/ \\ |

d e f

- List all answers generated by your definitions for the following

queries:

a. ?- sibling (X,Y).

X=b, Y=c;

X=c, Y=b;

X=d, Y=e;

X=e, Y=d.

b. ?- cousin(X,Y).

X=d, Y=f;

X=e, Y=f;

X=f, Y=d;

X=f, Y=e.

c. ?- grandchild(X,Y).

X=d, Y=a;

X=e, Y=a;

X=f, Y=a.

d. ?- descendent(X,Y).

X=b, Y=a;

X=c, Y=a;

X=d, Y=b;

X=e, Y=b;

X=f, Y=c;

X=d, Y=a;

X=e, Y=a;

X=f, Y=a.

Practical 8

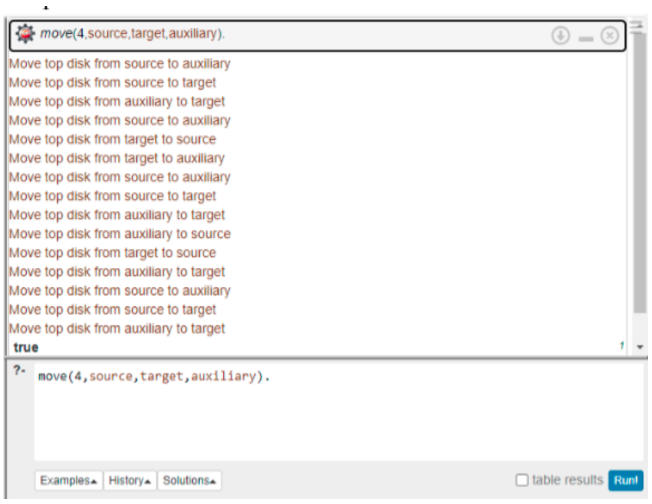
❖ **Aim:** Write a program to solve towers of Hanoi problem using Prolog.

Code:

```
move(1,X,Y,_) :-

write(&#39;Move top disk from &#39;), write(X), write(&#39; to &#39;),

write(Y), nl.

move(N,X,Y,Z) :-

N&gt;1,

M is N-1,

move(M,X,Z,Y),

move(1,X,Y,_),

move(M,Z,Y,X).
```

output:

?- move(4,source,target,auxiliary).



```
move(4,source,target,auxiliary).
Move top disk from source to auxiliary
Move top disk from source to target
Move top disk from auxiliary to target
Move top disk from source to auxiliary
Move top disk from target to source
Move top disk from target to auxiliary
Move top disk from source to auxiliary
Move top disk from source to target
Move top disk from auxiliary to target
Move top disk from auxiliary to source
Move top disk from target to source
Move top disk from auxiliary to target
Move top disk from source to auxiliary
Move top disk from source to target
Move top disk from auxiliary to target
true

?- move(4,source,target,auxiliary).
```

```
Move top disk from source to auxiliary
Move top disk from source to target
Move top disk from auxiliary to target
Move top disk from source to auxiliary
Move top disk from target to source
Move top disk from target to auxiliary
Move top disk from source to auxiliary
Move top disk from source to target
Move top disk from auxiliary to target
Move top disk from auxiliary to source
Move top disk from target to source
Move top disk from auxiliary to target
Move top disk from source to auxiliary
Move top disk from source to target
Move top disk from auxiliary to target
true ?
```

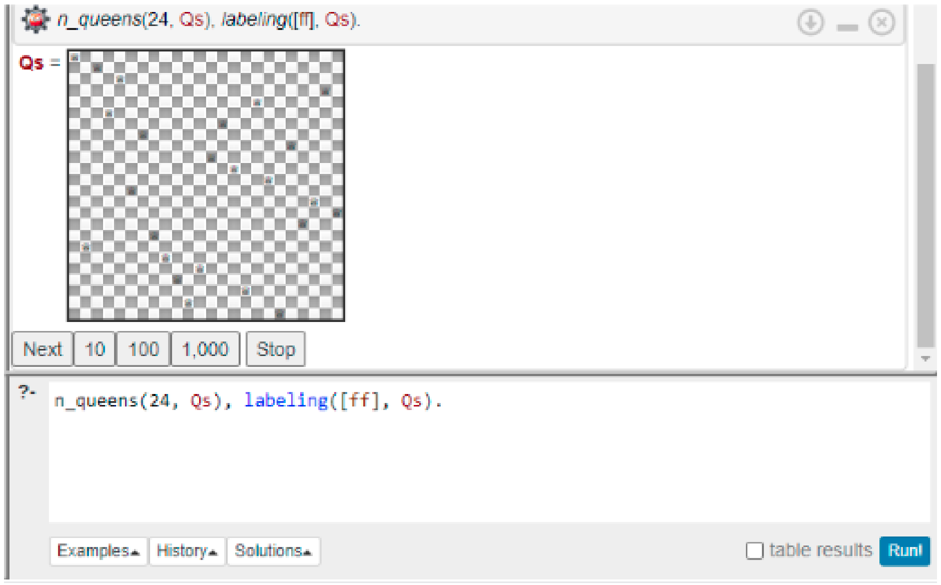
Practical 9

- **Aim:** Write a program to solve N-Queens problem using Prolog.
  Code:

```
% render solutions nicely.

:- use_rendering(chess).


%% n_queens(?N, ?Cols) is nondet.

%

% @param The k-th element of Cols is the column

number of the

% queen in row k.

% @author Markus Triska


:- use_module(library(clpfd)).


n_queens(N, Qs) :-

length(Qs, N),

Qs ins 1..N,

safe_queens(Qs).


safe_queens([]).

safe_queens([Q|Qs]) :-
safe_queens(Qs, Q, 1),


AI(3170716)Enrollment no.:181010107008
safe_queens(Qs).


safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
Q0 #\= Q,
abs(Q0 - Q) #\= D0,
D1 #= D0 + 1,
safe_queens(Qs, Q0, D1).
/** &lt;examples&gt;
?- n_queens(8, Qs), labeling([ff], Qs).
?- n_queens(24, Qs), labeling([ff], Qs).
?- n_queens(100, Qs), labeling([ff], Qs).


*/
```

Output:

1st

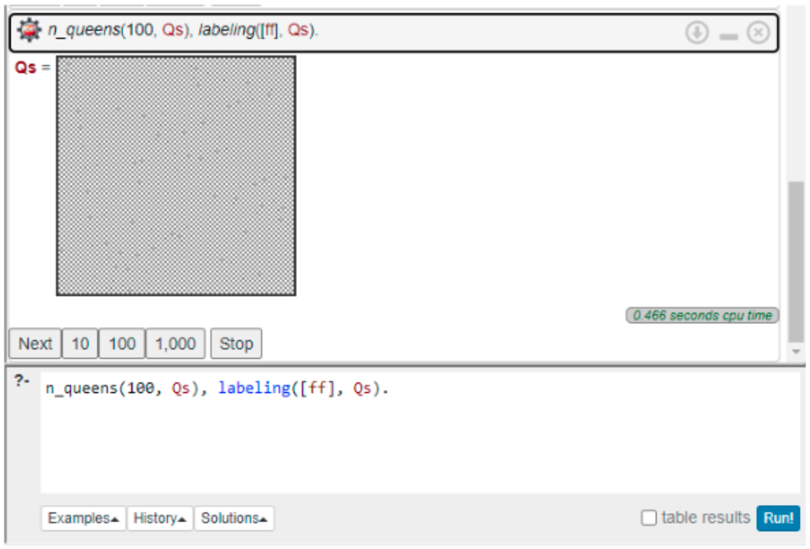


```
n_queens(24, Qs), labeling([ff], Qs).
Qs =
```

```
?- n_queens(24, Qs), labeling([ff], Qs).
```

2nd.

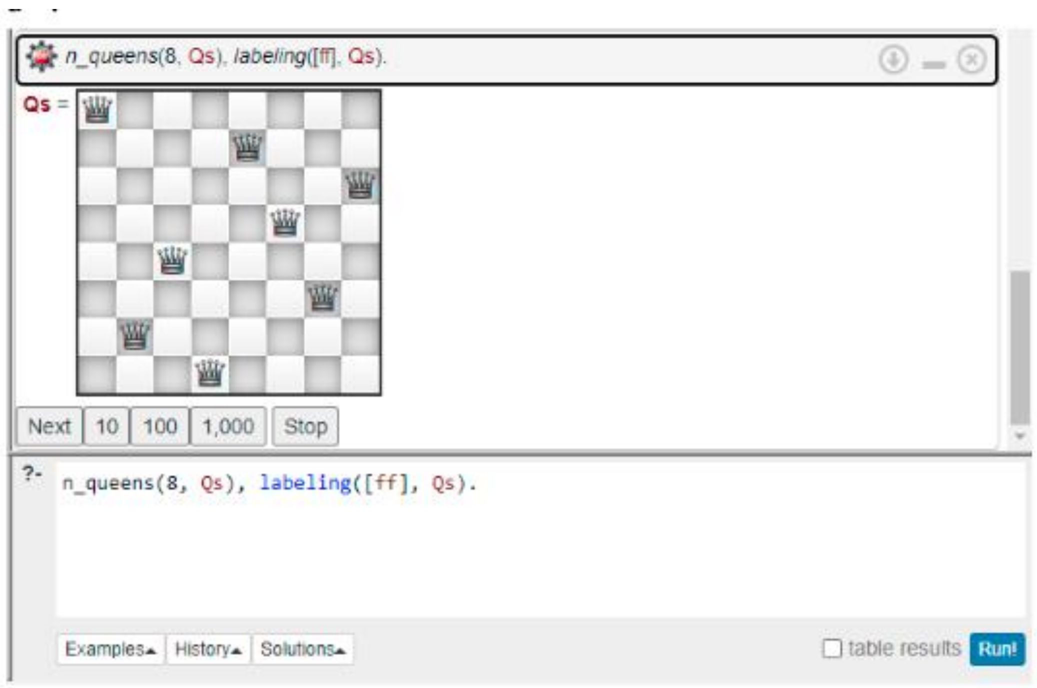
2nd



```
n_queens(100, Qs), labeling([ff], Qs).
Qs =
0.466 seconds cpu time
```

```
?- n_queens(100, Qs), labeling([ff], Qs).
```

3rd



```
n_queens(8, Qs), labeling([ff], Qs).
```

Practical 10

Aim: Write a program to solve 8 puzzle problem using Prolog.

Program:

```
/* 8_puzzle.pl */

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%

%%% A* Algorithm

%%%

%%%

%%% Nodes have form S#D#F#A

%%% where S describes the state or configuration

%%% D is the depth of the node

%%% F is the evaluation function value

%%% A is the ancestor list for the node


:- op(400,yfx,&#39;#&#39;). /* Node builder notation */


solve(State,Soln) :- f_function(State,0,F),


search([State#0#F#[]],S), reverse(S,Soln).
```

```
f_function(State,D,F) :- h_function(State,H),
F is D + H.

search([State#_#_#Soln|_], Soln) :- goal(State).
search([B|R],S) :- expand(B,Children),
insert_all(Children,R,Open),
search(Open,S).

insert_all([F|R],Open1,Open3) :- insert(F,Open1,Open2),
insert_all(R,Open2,Open3).
insert_all([],Open,Open).

insert(B,Open,Open) :- repeat_node(B,Open), ! .
insert(B,[C|R],[B,C|R]) :- cheaper(B,C), ! .
insert(B,[B1|R],[B1|S]) :- insert(B,R,S), !.
insert(B,[],[B]).

repeat_node(P#_#_#_, [P#_#_#_|_]).
cheaper( _#_#F1#_ , _#_#F2#_ ) :- F1 &lt; F2.

expand(State#D#_#S,All_My_Children) :-
bagof(Child#D1#F#[Move|S],
(D1 is D+1,

move(State,Child,Move),

f_function(Child,D1,F)),

All_My_Children).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%

%%% 8-puzzle solver

%%%

%%%

%%% State have form A/B/C/D/E/F/G/H/I

%%% where {A,...,I} = {0,...,8}

%%% 0 represents the empty tile

%%%

goal(1/2/3/8/0/4/7/6/5).

%%% The puzzle moves


left( A/0/C/D/E/F/H/I/J , 0/A/C/D/E/F/H/I/J ).

left( A/B/C/D/0/F/H/I/J , A/B/C/0/D/F/H/I/J ).

left( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/0/H/J ).

left( A/B/0/D/E/F/H/I/J , A/0/B/D/E/F/H/I/J ).
left( A/B/C/D/E/0/H/I/J , A/B/C/D/0/E/H/I/J ).
left( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/F/H/0/I ).

up( A/B/C/0/E/F/H/I/J , 0/B/C/A/E/F/H/I/J ).
up( A/B/C/D/0/F/H/I/J , A/0/C/D/B/F/H/I/J ).
up( A/B/C/D/E/0/H/I/J , A/B/0/D/E/C/H/I/J ).
up( A/B/C/D/E/F/0/I/J , A/B/C/0/E/F/D/I/J ).
up( A/B/C/D/E/F/H/0/J , A/B/C/D/0/F/H/E/J ).
up( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/0/H/I/F ).

right( A/0/C/D/E/F/H/I/J , A/C/0/D/E/F/H/I/J ).
right( A/B/C/D/0/F/H/I/J , A/B/C/D/F/0/H/I/J ).
right( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/H/J/0 ).
right( 0/B/C/D/E/F/H/I/J , B/0/C/D/E/F/H/I/J ).
right( A/B/C/0/E/F/H/I/J , A/B/C/E/0/F/H/I/J ).
right( A/B/C/D/E/F/0/I/J , A/B/C/D/E/F/I/0/J ).

down( A/B/C/0/E/F/H/I/J , A/B/C/H/E/F/0/I/J ).
down( A/B/C/D/0/F/H/I/J , A/B/C/D/I/F/H/0/J ).
down( A/B/C/D/E/0/H/I/J , A/B/C/D/E/J/H/I/0 ).
down( 0/B/C/D/E/F/H/I/J , D/B/C/0/E/F/H/I/J ).
down( A/0/C/D/E/F/H/I/J , A/E/C/D/0/F/H/I/J ).
down( A/B/0/D/E/F/H/I/J , A/B/F/D/E/0/H/I/J ).


%%% the heuristic function

h_function(Puzz,H) :- p_fcn(Puzz,P),

s_fcn(Puzz,S),

H is P + 3*S.


%%% the move

move(P,C,left) :- left(P,C).

move(P,C,up) :- up(P,C).

move(P,C,right) :- right(P,C).

move(P,C,down) :- down(P,C).


%%% the Manhattan distance function

p_fcn(A/B/C/D/E/F/G/H/I, P) :-

a(A,Pa), b(B,Pb), c(C,Pc),

d(D,Pd), e(E,Pe), f(F,Pf),

g(G,Pg), h(H,Ph), i(I,Pi),

P is Pa+Pb+Pc+Pd+Pe+Pf+Pg+Ph+Pg+Pi.


a(0,0). a(1,0). a(2,1). a(3,2). a(4,3). a(5,4). a(6,3). a(7,2). a(8,1).

b(0,0). b(1,1). b(2,0). b(3,1). b(4,2). b(5,3). b(6,2). b(7,3). b(8,2).
```

```
c(0,0). c(1,2). c(2,1). c(3,0). c(4,1). c(5,2). c(6,3). c(7,4). c(8,3).
d(0,0). d(1,1). d(2,2). d(3,3). d(4,2). d(5,3). d(6,2). d(7,2). d(8,0).
e(0,0). e(1,2). e(2,1). e(3,2). e(4,1). e(5,2). e(6,1). e(7,2). e(8,1).
f(0,0). f(1,3). f(2,2). f(3,1). f(4,0). f(5,1). f(6,2). f(7,3). f(8,2).
g(0,0). g(1,2). g(2,3). g(3,4). g(4,3). g(5,2). g(6,2). g(7,0). g(8,1).
h(0,0). h(1,3). h(2,3). h(3,3). h(4,2). h(5,1). h(6,0). h(7,1). h(8,2).
i(0,0). i(1,4). i(2,3). i(3,2). i(4,1). i(5,0). i(6,1). i(7,2). i(8,3).

%%% the out-of-cycle function
s_fcn(A/B/C/D/E/F/G/H/I, S) :-
s_aux(A,B,S1), s_aux(B,C,S2), s_aux(C,F,S3),



s_aux(F,I,S4), s_aux(I,H,S5), s_aux(H,G,S6),
s_aux(G,D,S7), s_aux(D,A,S8), s_aux(E,S9),
S is S1+S2+S3+S4+S5+S6+S7+S8+S9.

s_aux(0,0) :- !.
s_aux(_,1).

s_aux(X,Y,0) :- Y is X+1, !.
s_aux(8,1,0) :- !.
s_aux(_,_,2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%% 8-puzzle animation -- using VT100 character graphics
%%%
%%%
%%%

puzzle(P) :- solve(P,S),
animate(P,S),

message.

animate(P,S) :- initialize(P),
cursor(1,2), write(S),
cursor(1,22), write(&#39;Hit ENTER to step solver.&#39;),
get0(_X),
play_back(S).

:- dynamic location/3. %%% So that location of a tile
```

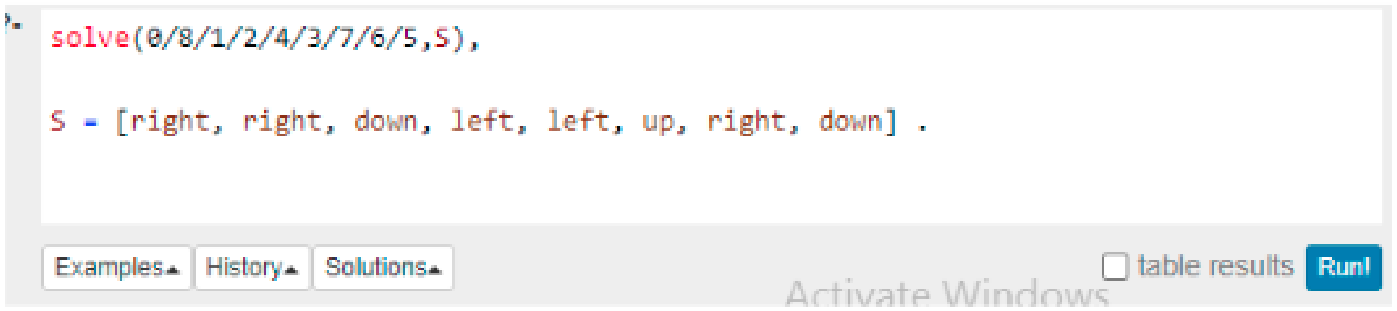
```
%%% can be retracted/asserted.

%%% Location(s) asserted and retracted

%%% by puzzle animator below


initialize(A/B/C/D/E/F/H/I/J) :-

cls,

retractall(location(_,_,_)),

assert(location(A,20,5)),

assert(location(B,30,5)),

assert(location(C,40,5)),

assert(location(F,40,10)),

assert(location(J,40,15)),


assert(location(I,30,15)),

assert(location(H,20,15)),

assert(location(D,20,10)),

assert(location(E,30,10)), draw_all.


draw_all :- draw(1), draw(2), draw(3), draw(4),

draw(5), draw(6), draw(7), draw(8).


%%% play_back([left,right,up,...]).
```

```
play_back([M|R]) :- call(M), get0(_X), play_back(R).

play_back([]) :- cursor(1,24). %%% Put cursor out of the way


message :- nl,nl,

write(&#39; ********************************************&#39;), nl,

write(&#39; * Enter 8-puzzle goals in the form ... *&#39;), nl,

write(&#39; * ?- puzzle(0/8/1/2/4/3/7/6/5). *&#39;), nl,

write(&#39; * Enter goal &#39;&#39;message&#39;&#39; to reread this.
*&#39;), nl,

write(&#39; ********************************************&#39;), nl,

nl.


cursor(X,Y) :- put(27), put(91), %%% ESC [

write(Y),

put(59), %%% ;

write(X),

put(72). %%% M


%%% clear the screen, quickly

cls :- put(27), put(&quot;[&quot;), put(&quot;2&quot;), put(&quot;J&quot;).
```

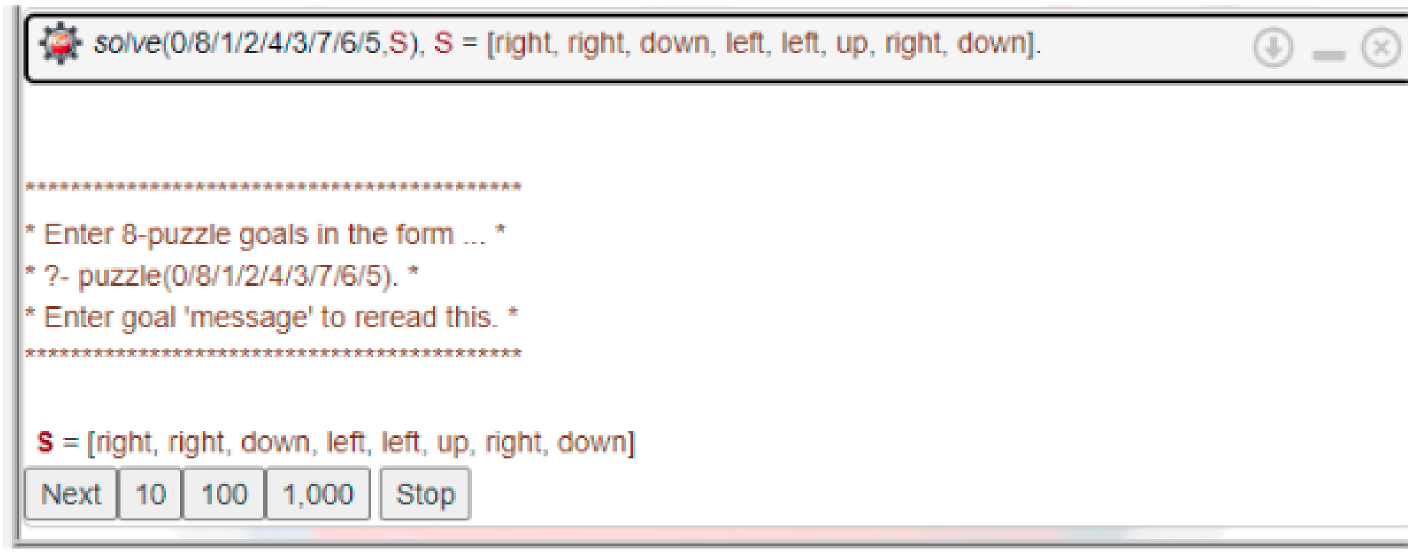
```
%%% video attributes -- bold and blink not working

plain :- put(27), put(&quot;[&quot;), put(&quot;0&quot;), put(&quot;m&quot;).

reverse_video :- put(27), put(&quot;[&quot;), put(&quot;7&quot;),
put(&quot;m&quot;).


%%% Tile objects, character map(s)

%%% Each tile should be drawn using the character map,

%%% drawn at &#39;location&#39;, which is asserted and retracted

%%% by &#39;playback&#39;.

character_map(N, [ [&#39; &#39;,&#39; &#39;,&#39; &#39;,&#39; &#39;,&#39;
&#39;,&#39; &#39;,&#39; &#39;],

[&#39; &#39;,&#39; &#39;,&#39; &#39;, N ,&#39; &#39;,&#39; &#39;,&#39;
&#39;],

[&#39; &#39;,&#39; &#39;,&#39; &#39;,&#39; &#39;,&#39; &#39;,&#39;
&#39;,&#39; &#39;] ]).



%%% move empty tile (spot) to the left

left :- retract(location(0,X0,Y0)),

Xnew is X0 - 10,

location(Tile,Xnew,Y0),

assert(location(0,Xnew,Y0)),
right(Tile),right(Tile),right(Tile),

right(Tile),right(Tile),

right(Tile),right(Tile),right(Tile),

right(Tile),right(Tile).


up :- retract(location(0,X0,Y0)),

Ynew is Y0 - 5,

location(Tile,X0,Ynew),

assert(location(0,X0,Ynew)),

down(Tile),down(Tile),down(Tile),down(Tile),down(Tile).


right :- retract(location(0,X0,Y0)),

Xnew is X0 + 10,

location(Tile,Xnew,Y0),


assert(location(0,Xnew,Y0)),

left(Tile),left(Tile),left(Tile),left(Tile),left(Tile),

left(Tile),left(Tile),left(Tile),left(Tile),left(Tile).


down :- retract(location(0,X0,Y0)),

Ynew is Y0 + 5,

location(Tile,X0,Ynew),
assert(location(0,X0,Ynew)),

up(Tile),up(Tile),up(Tile),up(Tile),up(Tile).


draw(Obj) :- reverse_video, character_map(Obj,M),

location(Obj,X,Y),

draw(X,Y,M), plain.


%%% hide tile

hide(Obj) :- character_map(Obj,M),

location(Obj,X,Y),

hide(X,Y,M).


hide(_,_,[]).

hide(X,Y,[R|G]) :- hide_row(X,Y,R),

Y1 is Y + 1,

hide(X,Y1,G).


hide_row(_,_,[]).

hide_row(X,Y,[_|R]) :- cursor(X,Y),

write(&#39; &#39;),

X1 is X + 1,

hide_row(X1,Y,R).
```

```
%%% draw tile
draw(_,_,[]).
draw(X,Y,[R|G]) :- draw_row(X,Y,R),
Y1 is Y + 1,
draw(X,Y1,G).

draw_row(_,_,[]).
draw_row(X,Y,[P|R]) :- cursor(X,Y),
write(P),
X1 is X + 1,
draw_row(X1,Y,R).
%%% Move an Object up
up(Obj) :- hide(Obj),
retract(location(Obj,X,Y)),
Y1 is Y - 1,
assert(location(Obj,X,Y1)),
draw(Obj).

down(Obj) :- hide(Obj),
retract(location(Obj,X,Y)),
Y1 is Y + 1,
assert(location(Obj,X,Y1)),

draw(Obj).


left(Obj) :- hide(Obj),

retract(location(Obj,X,Y)),

X1 is X - 1,

assert(location(Obj,X1,Y)),

draw(Obj).


right(Obj) :- hide(Obj),

retract(location(Obj,X,Y)),

X1 is X + 1,

assert(location(Obj,X1,Y)),

draw(Obj).


:- message.
```

Output:



```
?- solve(0/8/1/2/4/3/7/6/5,S),

S = [right, right, down, left, left, up, right, down] .
```



```
solve(0/8/1/2/4/3/7/6/5,S), S = [right, right, down, left, left, up, right, down].

*************************************************
* Enter 8-puzzle goals in the form ... *
* ?- puzzle(0/8/1/2/4/3/7/6/5). *
* Enter goal 'message' to reread this. *
*************************************************

S = [right, right, down, left, left, up, right, down]
```

Practical: 11

❖ **Aim:** Write a program to solve travelling salesman problem using Prolog.

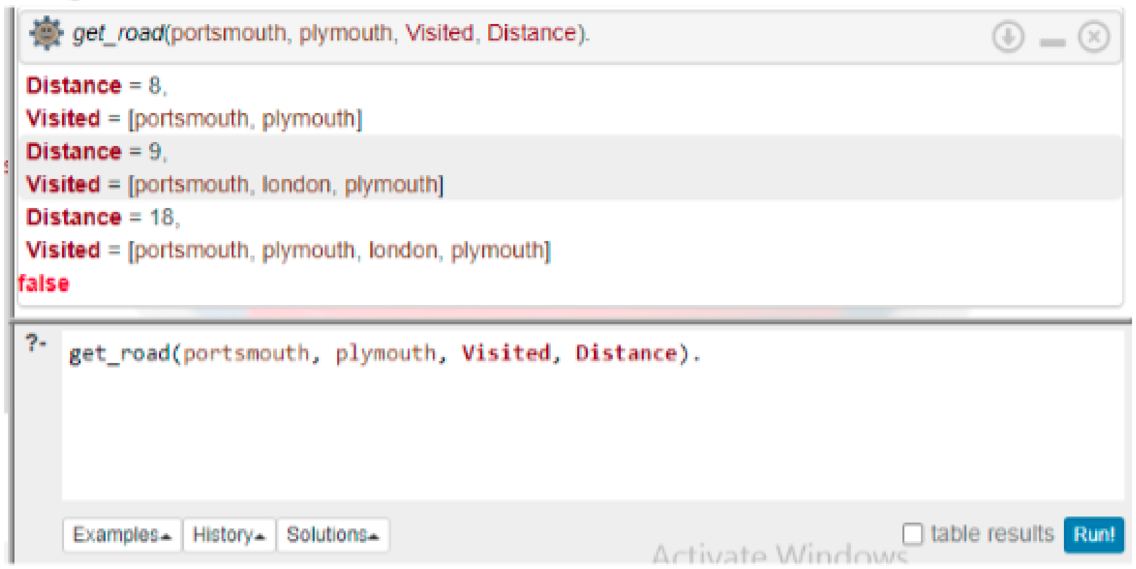
Program:

```
get_road(Start, End, Visited, Result) :-
get_road(Start, End, [Start], 0, Visited, Result).
get_road(Start, End, Waypoints, DistanceAcc, Visited,
TotalDistance) :-
road(Start, End, Distance),
reverse([End|Waypoints], Visited),
TotalDistance is DistanceAcc + Distance.
get_road(Start, End, Waypoints, DistanceAcc, Visited,
TotalDistance) :-
road(Start, Waypoint, Distance),
\+ member(Waypoint, Waypoints),
NewDistanceAcc is DistanceAcc + Distance,
get_road(Waypoint, End, [Waypoint|Waypoints],
NewDistanceAcc, Visited, TotalDistance).
road(birmingham,bristol, 9).
road(london,birmingham, 3).
road(london,bristol, 6).
road(london,plymouth, 5).
road(plymouth,london, 5).
```

road(portsmouth,london, 4).

road(portsmouth,plymouth, 8).

Output:



```
get_road(portsmouth, plymouth, Visited, Distance).
Distance = 8,
Visited = [portsmouth, plymouth]
Distance = 9,
Visited = [portsmouth, london, plymouth]
Distance = 18,
Visited = [portsmouth, plymouth, london, plymouth]
false

?- get_road(portsmouth, plymouth, Visited, Distance).
```