COLLEGE OF SCIENCE AND HUMANITIES
Ramapuram, Chennai.

Department of Computer Science

PRACTICAL RECORD

NAME :

REGISTER NO :

COURSE : **BSc**

SEMESTER / YEAR : **IV /II**

SUBJECT CODE : **UCS20D06J**

SUBJECT NAME : **Artificial Intelligence Lab**

APRIL 2022

FACULTY OF SCIENCE AND HUMANITIES

Ramapuram, Chennai.

Department of Computer Science

REGISTER NUMBER :

BONAFIDE CERTIFICATE

This is to certify that the bonafide work done by _____ in the subject **ARTIFICIAL INTELLIGENCE Lab [UCS20D06J] at** SRM IST, Ramapuram Campus in April 2022.

STAFF IN-CHARGE **HEAD OF THEDEPARTMENT**

Submitted for the University Practical Examination held, **SRM IST**, Ramapuram Campus on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

S. No	Date	NAME OF THE PROGRAMS	PAGE NO	SIGNATURE
1.		The Two Water Jug Puzzle		
2.		Solving Water Jug Problem using BFS & DFS		
3.		Route distance between two cities		
4.		Program for Tic Tac Toe game played by Single player against automated Computer player		
5.		Program for Tic Tac Toe game played by two different human players		
6.		Program to implement Tower of Hanoi		
7.		Program for building a magic square of Odd number of Rows and columns		
8.		Program for building a magic square of Even number of Rows and columns		
9.		Program to implement five House logic puzzle problem		
10.		Program for solving A* shortest path algorithm		
11.		Program which demonstrates Best First Search		
12.		Program to solve 8-Queens problem		
13.		Program which demonstrate the precedence properties of operators in C language		
14.		Program to calculate factorial of a number		
15.		Program to implement five House logic puzzle problem		

Ex No: 1 **Name:**

Date: **Reg No:**

The Two Water Jug Puzzle

Aim

Program showing the various possibilities involved in solving a water jug problem.

PROGRAM

```
// C++ program to count minimum number of steps
// required to measure d litres water using jugs
// of m liters and n liters capacity.
#include <bits/stdc++.h>
using namespace std;

// Utility function to return GCD of 'a'
// and 'b'.
int gcd(int a, int b)
{
        if (b==0)
        return a;
        return gcd(b, a%b);
}

/* fromCap -- Capacity of jug from which
                        water is poured
toCap -- Capacity of jug to which
                        water is poured
d        -- Amount to be measured */
int pour(int fromCap, int toCap, int d)
{
        // Initialize current amount of water
        // in source and destination jugs
        int from = fromCap;
        int to = 0;

        // Initialize count of steps required
        int step = 1; // Needed to fill "from" Jug

        // Break the loop when either of the two
        // jugs has d litre water
        while (from != d && to != d)
        {
                // Find the maximum amount that can be
                // poured
                int temp = min(from, toCap - to);

                // Pour "temp" liters from "from" to "to"
```

```
                to += temp;
                from -= temp;

                // Increment count of steps
                step++;

                if (from == d || to == d)
                        break;

                // If first jug becomes empty, fill it
                if (from == 0)
                {
                        from = fromCap;
                        step++;
                }

                // If second jug becomes full, empty it
                if (to == toCap)
                {
                        to = 0;
                        step++;
                }
        }
        return step;
}

// Returns count of minimum steps needed to
// measure d liter
int minSteps(int m, int n, int d)
{
        // To make sure that m is smaller than n
        if (m > n)
                swap(m, n);

        // For d > n we can't measure the water
        // using the jugs
        if (d > n)
                return -1;

        // If gcd of n and m does not divide d
        // then solution is not possible
        if ((d % gcd(n,m)) != 0)
                return -1;

        // Return minimum two cases:
        // a) Water of n liter jug is poured into
        // m liter jug
        // b) Vice versa of "a"
        return min(pour(n,m,d), // n to m
                        pour(m,n,d)); // m to n
}
// Driver code to test above
int main()
{
        int n = 3, m = 5, d = 4;

        printf("Minimum number of steps required is %d",
```

```
                minSteps(m, n, d));

        return 0;
}
```

OUTPUT

Minimum number of steps required is 6

RESULT

The program was successfully completed & executed.

Ex No: 2.(a) **Name:**

Date: **Reg No:**

Solving Water Jug Problem using BFS

Aim

Program for solving a water jug problem using Breadth first search and Depth first search (BFS & DFS).

PROGRAM

```
#include <bits/stdc++.h>
using namespace std;
class nodes{
        public:
                pair<int,int> p;
                int first;
                int second;
                string s;
};
string makestring(int a,int b){
        std::stringstream out1;
        std::stringstream out2;
        string t1,t2,str;
    out1 << a;
    t1 = out1.str();
    out2 << b;
    t2 = out2.str();
    str = "("+t1+","+t2+")";
    return str;
}
int main()
{
        int counter = 0;
    ios::sync_with_stdio(false);
    //pair<int,int> cap,ini,final;
    nodes cap,ini,final;
    ini.p.first=0,ini.p.second=0;
    ini.s = makestring(ini.p.first,ini.p.second);
    //Input initial values
    cout<<"Enter the capacity of 2 jugs\n";
    cin>>cap.p.first>>cap.p.second;
    //input final values
    cout<<"Enter the required jug config\n";
    cin>>final.p.first>>final.p.second;
    //Using BFS to find the answer
    queue<nodes> q;
    q.push(ini);
    nodes jug;
    while(!q.empty()){
        //Base case
        jug = q.front();
```

```
        if(jug.p.first == final.p.first){// && jug.p.second ==
final.p.second){
                cout<<jug.s<<endl;
                // counter++;
                // if(counter==5)
                        return 0;
        }
        nodes temp = jug;
        //Fill 1st Jug
        if(jug.p.first<cap.p.first){
                        temp.p = make_pair(cap.p.first,jug.p.second);
                        temp.s = jug.s +
makestring(temp.p.first,temp.p.second);
                        q.push(temp);
        }
        //Fill 2nd Jug
        if(jug.p.second<cap.p.second){
                        temp.p = make_pair(jug.p.first,cap.p.second);
                        temp.s = jug.s +
makestring(temp.p.first,temp.p.second);
                        q.push(temp);
        }
        //Empty 1st Jug
        if(jug.p.first>0){
                        temp.p = make_pair(0,jug.p.second);
                        temp.s = jug.s +
makestring(temp.p.first,temp.p.second);
                        q.push(temp);
        }
        //Empty 2nd Jug
        if(jug.p.second>0){
                        temp.p = make_pair(jug.p.first,0);
                        temp.s = jug.s +
makestring(temp.p.first,temp.p.second);
                        q.push(temp);
        }
        //Pour from 1st jug to 2nd until its full
        if(jug.p.first>0 && (jug.p.first+jug.p.second)>=cap.p.second){
                temp.p = make_pair((jug.p.first-(cap.p.second-
jug.p.second)),cap.p.second);
                temp.s = jug.s + makestring(temp.p.first,temp.p.second);
                q.push(temp);
        }
        //Pour from 2nd jug to 1st until its full
        if(jug.p.second>0 && (jug.p.first+jug.p.second)>=cap.p.first){
                temp.p = make_pair(cap.p.first,(jug.p.second-(cap.p.first-
jug.p.first)));
                temp.s = jug.s + makestring(temp.p.first,temp.p.second);
                q.push(temp);
        }
        //Pour all water from 1st to 2nd
        if(jug.p.first>0 && (jug.p.first+jug.p.second)<=cap.p.second){
                temp.p = make_pair(0,jug.p.first+jug.p.second);
                temp.s = jug.s + makestring(temp.p.first,temp.p.second);
                q.push(temp);
        }
        //Pour from 2nd jug to 1st until its full
```

```
        if(jug.p.second>0 && (jug.p.first+jug.p.second)<=cap.p.first){
                temp.p = make_pair(jug.p.first+jug.p.second,0);
                temp.s = jug.s + makestring(temp.p.first,temp.p.second);
                q.push(temp);
        }
        q.pop();
    }
        return 0;
}
```

OUTPUT

Enter the capacity of 2 jugs

4

3

Enter the required jug config

2

0

(0,0)(4,0)(1,3)(1,0)(0,1)(4,1)(2,3)

Result

The program was successfully completed & executed.

Ex No: 2.(b) **Name:**

Date: **Reg No:**

Solving Water Jug Problem using DFS

Aim

Program to Solve Water Jug Problem using DFS

PROGRAM

```
#include <cstdio>
#include <stack>
#include <map>
#include <algorithm>
using namespace std;

// Representation of a state (x, y)
// x and y are the amounts of water in litres in the two jugs respectively
struct state {
    int x, y;

    // Used by map to efficiently implement lookup of seen states
    bool operator < (const state& that) const {
        if (x != that.x) return x < that.x;
        return y < that.y;
    }
};

// Capacities of the two jugs respectively and the target amount
int capacity_x, capacity_y, target;

void dfs(state start, stack <pair <state, int> >& path)   {
    stack <state> s;
    state goal = (state) {-1, -1};

    // Stores seen states so that they are not revisited and
    // maintains their parent states for finding a path through
    // the state space
    // Mapping from a state to its parent state and rule no. that
    // led to this state
    map <state, pair <state, int> > parentOf;

    s.push(start);
    parentOf[start] = make_pair(start, 0);

    while (!s.empty())    {
        // Get the state at the front of the stack
        state top = s.top();
        s.pop();

        // If the target state has been found, break
        if (top.x == target || top.y == target) {
```

```
            goal = top;
            break;
        }

        // Find the successors of this state
        // This step uses production rules to produce successors of the
current state
        // while pruning away branches which have been seen before

        // Rule 1: (x, y) -> (capacity_x, y) if x < capacity_x
        // Fill the first jug
        if (top.x < capacity_x)  {
            state child = (state) {capacity_x, top.y};
            // Consider this state for visiting only if it has not been
visited before
            if (parentOf.find(child) == parentOf.end()) {
                s.push(child);
                parentOf[child] = make_pair(top, 1);
            }
        }

        // Rule 2: (x, y) -> (x, capacity_y) if y < capacity_y
        // Fill the second jug
        if (top.y < capacity_y)  {
            state child = (state) {top.x, capacity_y};
            if (parentOf.find(child) == parentOf.end()) {
                s.push(child);
                parentOf[child] = make_pair(top, 2);
            }
        }

        // Rule 3: (x, y) -> (0, y) if x > 0
        // Empty the first jug
        if (top.x > 0)  {
            state child = (state) {0, top.y};
            if (parentOf.find(child) == parentOf.end()) {
                s.push(child);
                parentOf[child] = make_pair(top, 3);
            }
        }

        // Rule 4: (x, y) -> (x, 0) if y > 0
        // Empty the second jug
        if (top.y > 0)  {
            state child = (state) {top.x, 0};
            if (parentOf.find(child) == parentOf.end()) {
                s.push(child);
                parentOf[child] = make_pair(top, 4);
            }
        }

        // Rule 5: (x, y) -> (min(x + y, capacity_x), max(0, x + y -
capacity_x)) if y > 0
        // Pour water from the second jug into the first jug until the first
jug is full
        // or the second jug is empty
        if (top.y > 0)  {
```

```
            state child = (state) {min(top.x + top.y, capacity_x), max(0,
top.x + top.y - capacity_x)};
            if (parentOf.find(child) == parentOf.end()) {
                s.push(child);
                parentOf[child] = make_pair(top, 5);
            }
        }

        // Rule 6: (x, y) -> (max(0, x + y - capacity_y), min(x + y,
capacity_y)) if x > 0
        // Pour water from the first jug into the second jug until the second
jug is full
        // or the first jug is empty
        if (top.x > 0)  {
            state child = (state) {max(0, top.x + top.y - capacity_y),
min(top.x + top.y, capacity_y)};
            if (parentOf.find(child) == parentOf.end()) {
                s.push(child);
                parentOf[child] = make_pair(top, 6);
            }
        }
    }

    // Target state was not found
    if (goal.x == -1 || goal.y == -1)
        return;

    // backtrack to generate the path through the state space
    path.push(make_pair(goal, 0));
    // remember parentOf[start] = (start, 0)
    while (parentOf[path.top().first].second != 0)
        path.push(parentOf[path.top().first]);
}

int main()  {
    stack <pair <state, int> > path;

    printf("Enter the capacities of the two jugs : ");
    scanf("%d %d", &capacity_x, &capacity_y);
    printf("Enter the target amount : ");
    scanf("%d", &target);

    dfs((state) {0, 0}, path);
    if (path.empty())
        printf("\nTarget cannot be reached.\n");
    else    {
        printf("\nNumber of moves to reach the target : %d\nOne path to the
target is as follows :\n", path.size() - 1);
        while (!path.empty())   {
            state top = path.top().first;
            int rule = path.top().second;
            path.pop();

            switch (rule)   {
                case 0: printf("State : (%d, %d)\n#\n", top.x, top.y);
                        break;
                case 1: printf("State : (%d, %d)\nAction : Fill the first
```

```
jug\n", top.x, top.y);
                        break;
                case 2: printf("State : (%d, %d)\nAction : Fill the second
jug\n", top.x, top.y);
                        break;
                case 3: printf("State : (%d, %d)\nAction : Empty the first
jug\n", top.x, top.y);
                        break;
                case 4: printf("State : (%d, %d)\nAction : Empty the second
jug\n", top.x, top.y);
                        break;
                case 5: printf("State : (%d, %d)\nAction : Pour from second
jug into first jug\n", top.x, top.y);
                        break;
                case 6: printf("State : (%d, %d)\nAction : Pour from first
jug into second jug\n", top.x, top.y);
                        break;
            }
        }
    }

    return 0;
}
```

OUTPUT

Enter the capacities of the two jugs : 4
3
Enter the target amount : 2
Number of moves to reach the target : 4
One path to the target is as follows :
State : (0, 0)
Action : Fill the second jug
State : (0, 3)
Action : Pour from second jug into first jug
State : (3, 0)
Action : Fill the second jug
State : (3, 3)
Action : Pour from second jug into first jug
State : (4, 2)

Result

The program was successfully completed & executed.

Ex No: 3 **Name:**

Date: **Reg No:**

Route distance between two cities

Aim

Program to find out route distance between two cities.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()
{
int G[MAX][MAX],i,j,n,u;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
printf("\nEnter the starting node:");
scanf("%d",&u);
dijkstra(G,n,u);
return 0;
}

void dijkstra(int G[MAX][MAX],int n,int startnode)
{

int cost[MAX][MAX],distance[MAX],pred[MAX];
int visited[MAX],count,mindistance,nextnode,i,j;
//pred[] stores the predecessor of each node
//count gives the number of nodes seen so far
//create the cost matrix
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(G[i][j]==0)
cost[i][j]=INFINITY;
else
cost[i][j]=G[i][j];
//initialize pred[],distance[] and visited[]
for(i=0;i<n;i++)
{
distance[i]=cost[startnode][i];
pred[i]=startnode;
visited[i]=0;
}
distance[startnode]=0;
```

```
visited[startnode]=1;
count=1;
while(count<n-1)
{
mindistance=INFINITY;
//nextnode gives the node at minimum distance
for(i=0;i<n;i++)
if(distance[i]<mindistance&&!visited[i])
{
mindistance=distance[i];
nextnode=i;
}
//check if a better path exists through nextnode
visited[nextnode]=1;
for(i=0;i<n;i++)
if(!visited[i])
if(mindistance+cost[nextnode][i]<distance[i])
{
distance[i]=mindistance+cost[nextnode][i];
pred[i]=nextnode;
}
count++;
}

//print the path and distance of each node
for(i=0;i<n;i++)
if(i!=startnode)
{
printf("\nDistance of node%d=%d",i,distance[i]);
printf("\nPath=%d",i);
j=i;
do
{
j=pred[j];
printf("<-%d",j);
}while(j!=startnode);
}
}
```

OUTPUT

```
Enter no. of vertices:5
Enter the adjacency matrix:
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0

Enter the starting node:0

Distance of node1=10
Path=1<-0
Distance of node2=50
Path=2<-3<-0
Distance of node3=30
Path=3<-0
Distance of node4=60
Path=4<-2<-3<-0
```

Result

The program was successfully completed & executed.

Ex No: 4 **Name:**

Date: **Reg No:**

Program for Tic Tac Toe game played by Single player against automated Computer player

Aim

Program for Tic Tac Toe game played by Single player against automated Computer player.

PROGRAM

```
// A C++ Program to play tic-tac-toe

#include<bits/stdc++.h>
using namespace std;

#define COMPUTER 1
#define HUMAN 2

#define SIDE 3 // Length of the board

// Computer will move with 'O'
// and human with 'X'
#define COMPUTERMOVE 'O'
#define HUMANMOVE 'X'

// A function to show the current board status
void showBoard(char board[][SIDE])
{
        printf("\n\n");

        printf("\t\t\t %c | %c | %c \n", board[0][0],
                                                board[0][1],
board[0][2]);
        printf("\t\t\t--------------\n");
        printf("\t\t\t %c | %c | %c \n", board[1][0],
                                                board[1][1],
board[1][2]);
        printf("\t\t\t--------------\n");
        printf("\t\t\t %c | %c | %c \n\n", board[2][0],
                                                board[2][1],
board[2][2]);

        return;
}

// A function to show the instructions
void showInstructions()
{
        printf("\t\t\t Tic-Tac-Toe\n\n");
        printf("Choose a cell numbered from 1 to 9 as below"
                        " and play\n\n");
```

```
        printf("\t\t\t 1 | 2 | 3 \n");
        printf("\t\t\t--------------\n");
        printf("\t\t\t 4 | 5 | 6 \n");
        printf("\t\t\t--------------\n");
        printf("\t\t\t 7 | 8 | 9 \n\n");

        printf("-\t-\t-\t-\t-\t-\t-\t-\t-\n\n");

        return;
}


// A function to initialise the game
void initialise(char board[][SIDE], int moves[])
{
        // Initiate the random number generator so that
        // the same configuration doesn't arises
        srand(time(NULL));

        // Initially the board is empty
        for (int i=0; i<SIDE; i++)
        {
                for (int j=0; j<SIDE; j++)
                        board[i][j] = ' ';
        }

        // Fill the moves with numbers
        for (int i=0; i<SIDE*SIDE; i++)
                moves[i] = i;

        // randomise the moves
        random_shuffle(moves, moves + SIDE*SIDE);

        return;
}

// A function to declare the winner of the game
void declareWinner(int whoseTurn)
{
        if (whoseTurn == COMPUTER)
                printf("COMPUTER has won\n");
        else
                printf("HUMAN has won\n");
        return;
}

// A function that returns true if any of the row
// is crossed with the same player's move
bool rowCrossed(char board[][SIDE])
{
        for (int i=0; i<SIDE; i++)
        {
                if (board[i][0] == board[i][1] &&
                        board[i][1] == board[i][2] &&
                        board[i][0] != ' ')
                        return (true);
        }
```

```
        return(false);
}
// A function that returns true if any of the column
// is crossed with the same player's move
bool columnCrossed(char board[][SIDE])
{
        for (int i=0; i<SIDE; i++)
        {
                if (board[0][i] == board[1][i] &&
                        board[1][i] == board[2][i] &&
                        board[0][i] != ' ')
                        return (true);
        }
        return(false);
}
// A function that returns true if any of the diagonal
// is crossed with the same player's move
bool diagonalCrossed(char board[][SIDE])
{
        if (board[0][0] == board[1][1] &&
                board[1][1] == board[2][2] &&
                board[0][0] != ' ')
                return(true);

        if (board[0][2] == board[1][1] &&
                board[1][1] == board[2][0] &&
                board[0][2] != ' ')
                return(true);

        return(false);
}

// A function that returns true if the game is over
// else it returns a false
bool gameOver(char board[][SIDE])
{
        return(rowCrossed(board) || columnCrossed(board)
                        || diagonalCrossed(board) );
}

// A function to play Tic-Tac-Toe
void playTicTacToe(int whoseTurn)
{
        // A 3*3 Tic-Tac-Toe board for playing
        char board[SIDE][SIDE];

        int moves[SIDE*SIDE];

        // Initialise the game
        initialise(board, moves);

        // Show the instructions before playing
        showInstructions();

        int moveIndex = 0, x, y;

        // Keep playing till the game is over or it is a draw
```

```
        while (gameOver(board) == false &&
                        moveIndex != SIDE*SIDE)
        {
                if (whoseTurn == COMPUTER)
                {
                        x = moves[moveIndex] / SIDE;
                        y = moves[moveIndex] % SIDE;
                        board[x][y] = COMPUTERMOVE;
                        printf("COMPUTER has put a %c in cell %d\n",
                                        COMPUTERMOVE, moves[moveIndex]+1);
                        showBoard(board);
                        moveIndex ++;
                        whoseTurn = HUMAN;
                }

                else if (whoseTurn == HUMAN)
                {
                        x = moves[moveIndex] / SIDE;
                        y = moves[moveIndex] % SIDE;
                        board[x][y] = HUMANMOVE;
                        printf ("HUMAN has put a %c in cell %d\n",
                                        HUMANMOVE, moves[moveIndex]+1);
                        showBoard(board);
                        moveIndex ++;
                        whoseTurn = COMPUTER;
                }
        }

        // If the game has drawn
        if (gameOver(board) == false &&
                        moveIndex == SIDE * SIDE)
                printf("It's a draw\n");
        else
        {
                // Toggling the user to declare the actual
                // winner
                if (whoseTurn == COMPUTER)
                        whoseTurn = HUMAN;
                else if (whoseTurn == HUMAN)
                        whoseTurn = COMPUTER;

                // Declare the winner
                declareWinner(whoseTurn);
        }
        return;
}
// Driver program
int main()
{
        // Let us play the game with COMPUTER starting first
        playTicTacToe(COMPUTER);

        return (0);
}
```

OUTPUT

```
Tic-Tac-Toe
Choose a cell numbered from 1 to 9 as below and play
1 | 2 | 3
---------
4 | 5 | 6
---------
7 | 8 | 9
- - - -
- - - -
- - - -
COMPUTER has put a O in cell 9
| |
---------
| |
---------
| | O

HUMAN has put a X in cell 5
| |
---------
| X |
---------
|
| O

COMPUTER has put a O in cell 4

|
|
---
-----------
O
| X |
---
-----------
|
| O
HUMAN has put a X in cell 3

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

COMPUTER has put a O in cell 1

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

HUMAN has put a X in cell 7

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

HUMAN has won
/tmp/cA9ekskY9t.o
Tic-Tac-Toe

Choose a cell numbered from 1 to 9 as below and play

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

- - - -
- - - -
- - - -
COMPUTER has put a O in cell 9

|
|
---
-----------
| |
---
-----------
|
| O

HUMAN has put a X in cell 6

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

COMPUTER has put a O in cell 4

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

HUMAN has put a X in cell 5

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

COMPUTER has put a O in cell 7

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

HUMAN has put a X in cell 8

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

COMPUTER has put a O in cell 2

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

HUMAN has put a X in cell 1

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

COMPUTER has put a O in cell 3

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

It's a draw
```

Result

The program was successfully completed & executed.

Ex No: 5 **Name:**

Date: **Reg No:**

Program for Tic Tac Toe game played by two different human players

Aim

Program for Tic Tac Toe game played by two different human players.

PROGRAM

```
#include <iostream>
using namespace std;

char board[3][3] = {{'1', '2', '3'}, {'4', '5', '6'}, {'7', '8', '9'}};
static int turnnumber = 1 ;
bool winner = false, flag = false ;
bool win(){
     if(board[0][0]==board[1][1]&&board[1][1]==board[2][2])
    winner = true ;
    if(board[0][2]==board[1][1]&&board[1][1]==board[2][0])
    winner = true ;
if(board[1][0]==board[1][1]&&board[1][1]==board[1][2])
    winner = true ;
if(board[0][0]==board[0][1]&&board[0][1]==board[0][2])
    winner = true ;
if(board[2][0]==board[2][1]&&board[2][1]==board[2][2])
    winner = true ;
if(board[0][0]==board[1][0]&&board[1][0]==board[2][0])
    winner = true ;
if(board[0][1]==board[1][1]&&board[1][1]==board[2][1])
    winner = true ;
if(board[0][2]==board[1][2]&&board[1][2]==board[2][2])
    winner = true ;

    if(winner==true&&turnnumber==1)
       cout << "player2 won \n\n" ;
    if(winner==true&&turnnumber==2)
       cout << "player1 won \n\n" ;

    return winner;
}

void view()
{
    for (int i = 0; i < 3; i++)
    {
        for (int x = 0; x < 3; x++)
        {
            cout << "[" << board[i][x] << "]  ";
        }
```

```
        cout << endl
             << "-------------" << endl;
    }
}

void players()
{

    char player1 = 'X', player2 = 'O';
    int number;
    cout << "\nplayer " << turnnumber << " it's your turn ";
    if(turnnumber==1)
       turnnumber++;
    else if(turnnumber==2)
       turnnumber--;
    char player;
    if(turnnumber==1)
       player=player2;
    if(turnnumber==2)
       player=player1;
    cin >> number ;

    switch(number){

    case 1:
        board[0][0] = player;
        break;
    case 2:
        board[0][1] = player;
        break;
    case 3:
        board[0][2] = player;
        break;
    case 4:
        board[1][0] = player;
        break;
    case 5:
        board[1][1] = player;
        break;
    case 6:
        board[1][2] = player;
        break;
    case 7:
        board[2][0] = player;
        break;
    case 8:
        board[2][1] = player;
        break;
    case 9:
        board[2][2] = player;
        break;
    default:
        cout << "\nwrong number\n";
        players();

    }
    system("cls");
```

```
    view();

    if(!win())
        players();
}

int main()
{
    view();
    players();
}
```

OUTPUT

```
[1]  [2]  [3]
-------------
[4]  [5]  [6]
-------------
[7]  [8]  [9]
-------------

player 1 it's your turn 1
sh: 1: cls: not found
[X]  [2]  [3]
-------------
[4]  [5]  [6]
-------------
[7]  [8]  [9]
-------------

player 2 it's your turn 4
sh: 1: cls: not found
[X]  [2]  [3]
-------------
[O]  [5]  [6]
-------------
[7]  [8]  [9]
-------------
```

```
player 1 it's your turn 5
sh: 1: cls: not found
[X]  [2]  [3]
-------------
[O]  [X]  [6]
-------------
[7]  [8]  [9]
-------------

player 2 it's your turn 7
sh: 1: cls: not found
[X]  [2]  [3]
-------------
[O]  [X]  [6]
-------------
[O]  [8]  [9]
-------------
```

```
player 1 it's your turn 9
sh: 1: cls: not found
[X]  [2]  [3]
-------------
[O]  [X]  [6]
-------------
[O]  [8]  [X]
-------------
player1 won
```

Result

The program was successfully completed & executed.

Ex No: 6 **Name:**

Date: **Reg No:**

Program to implement Tower of Hanoi

Aim

Program for to implement Tower of Hanoi

PROGRAM

```
#include<iostream>
using namespace std;

//tower of HANOI function implementation
void TOH(int n,char Sour, char Aux,char Des)
{
        if(n==1)
        {
                cout<<"Move Disk "<<n<<" from "<<Sour<<" to "<<Des<<endl;
                return;
        }

        TOH(n-1,Sour,Des,Aux);
        cout<<"Move Disk "<<n<<" from "<<Sour<<" to "<<Des<<endl;
        TOH(n-1,Aux,Sour,Des);
}

//main program
int main()
{
        int n;

        cout<<"Enter no. of disks:";
        cin>>n;
        //calling the TOH
        TOH(n,'A','B','C');

        return 0;
}
```

OUTPUT

```
Enter no. of disks:3
Move Disk 1 from A to C
Move Disk 2 from A to B
Move Disk 1 from C to B
Move Disk 3 from A to C
Move Disk 1 from B to A
Move Disk 2 from B to C
Move Disk 1 from A to C
```

Result

The program was successfully completed & executed.

Ex No: 7 **Name:**

Date: **Reg No:**

Program for building a magic square of Odd number of Rows and columns.

Aim

Program for building a magic square of Odd number of Rows and columns.

PROGRAM

```
// C++ program to generate odd sized magic squares
#include <bits/stdc++.h>
using namespace std;

// A function to generate odd sized magic squares
void generateSquare(int n)
{
	int magicSquare[n][n];

	// set all slots as 0
	memset(magicSquare, 0, sizeof(magicSquare));

	// Initialize position for 1
	int i = n / 2;
	int j = n - 1;

	// One by one put all values in magic square
	for (int num = 1; num <= n * n;) {
		if (i == -1 && j == n) // 3rd condition
		{
			j = n - 2;
			i = 0;
		}
		else {
			// 1st condition helper if next number
			// goes to out of square's right side
			if (j == n)
				j = 0;

			// 1st condition helper if next number
			// is goes to out of square's upper side
			if (i < 0)
				i = n - 1;
		}
		if (magicSquare[i][j]) // 2nd condition
		{
			j -= 2;
			i++;
			continue;
		}
```

```
                else
                        magicSquare[i][j] = num++; // set number

                j++;
                i--; // 1st condition
        }

        // Print magic square
        cout << "The Magic Square for n=" << n
                << ":\nSum of "
                        "each row or column "
                << n * (n * n + 1) / 2 << ":\n\n";
        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++)

                        // setw(7) is used so that the matrix gets
                        // printed in a proper square fashion.
                        cout << setw(4) << magicSquare[i][j] << " ";
                cout << endl;
        }
}

// Driver code
int main()
{

        // Works only when n is odd
        int n = 7;
        generateSquare(n);
        return 0;
}

// This code is contributed by rathbhupendra
```

OUTPUT:

The Magic Square for n=7:
Sum of each row or column 175:

```
 20  12   4  45  37  29  28
 11   3  44  36  35  27  19
  2  43  42  34  26  18  10
 49  41  33  25  17   9   1
 40  32  24  16   8   7  48
 31  23  15  14   6  47  39
 22  21  13   5  46  38  30
```

Result

The program was successfully completed & executed.

Ex No: 8 **Name:**

Date: **Reg No:**

Program for building a magic square of Even number of Rows and columns.

Aim

Program for building a magic square of Even number of Rows and columns.

PROGRAM

```
// C++ program to generate odd sized magic squares
#include <bits/stdc++.h>
using namespace std;

// A function to generate odd sized magic squares
void generateSquare(int n)
{
        int magicSquare[n][n];

        // set all slots as 0
        memset(magicSquare, 0, sizeof(magicSquare));

        // Initialize position for 1
        int i = n / 2;
        int j = n - 1;

        // One by one put all values in magic square
        for (int num = 1; num <= n * n;) {
                if (i == -1 && j == n) // 3rd condition
                {
                        j = n - 2;
                        i = 0;
                }
                else {
                        // 1st condition helper if next number
                        // goes to out of square's right side
                        if (j == n)
                                j = 0;

                        // 1st condition helper if next number
                        // is goes to out of square's upper side
                        if (i < 0)
                                i = n - 1;
                }
                if (magicSquare[i][j]) // 2nd condition
                {
                        j -= 2;
                        i++;
                        continue;
                }
```

```
                else
                        magicSquare[i][j] = num++; // set number

                j++;
                i--; // 1st condition
        }

        // Print magic square
        cout << "The Magic Square for n=" << n
                << ":\nSum of "
                        "each row or column "
                << n * (n * n + 1) / 2 << ":\n\n";
        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++)

                        // setw(7) is used so that the matrix gets
                        // printed in a proper square fashion.
                        cout << setw(4) << magicSquare[i][j] << " ";
                cout << endl;
        }
}

// Driver code
int main()
{

        // Works only when n is odd
        int n = 7;
        generateSquare(n);
        return 0;
}
```

OUTPUT:

The Magic Square for n=7:
Sum of each row or column 175:

```
 20  12   4  45  37  29  28
 11   3  44  36  35  27  19
 2  43  42  34  26  18  10
 49  41  33  25  17   9   1
 40  32  24  16   8   7  48
 31  23  15  14   6  47  39
 22  21  13   5  46  38  30
```

Result

The program was successfully completed & executed.

Ex No: 9 **Name:**

Date: **Reg No:**

Program to implement five House logic puzzle problem

Aim

Program for to implement five House logic puzzle problem

PROGRAM

```
#include <stdio.h>
#include <string.h>
enum HouseStatus { Invalid, Underfull, Valid };
enum Attrib { C, M, D, A, S };
// Unfilled attributes are represented by -1
enum Colors { Red, Green, White, Yellow, Blue };
enum Mans { English, Swede, Dane, German, Norwegian };
enum Drinks { Tea, Coffee, Milk, Beer, Water };
enum Animals { Dog, Birds, Cats, Horse, Zebra };
enum Smokes { PallMall, Dunhill, Blend, BlueMaster, Prince };
void printHouses(int ha[5][5]) {
const char *color[] = { "Red", "Green", "White", "Yellow", "Blue" };
const char *man[] = { "English", "Swede", "Dane", "German", "Norwegian" };
const char *drink[] = { "Tea", "Coffee", "Milk", "Beer", "Water" };
const char *animal[] = { "Dog", "Birds", "Cats", "Horse", "Zebra" };
const char *smoke[] = { "PallMall", "Dunhill", "Blend", "BlueMaster",
"Prince" };
printf("%-10.10s%-10.10s%-10.10s%-10.10s%-10.10s%-10.10s\n",

"House", "Color", "Man", "Drink", "Animal", "Smoke");
for (int i = 0; i < 5; i++) {
printf("%-10d", i);
if (ha[i][C] >= 0)
printf("%-10.10s", color[ha[i][C]]);
else
printf("%-10.10s", "-");
if (ha[i][M] >= 0)
printf("%-10.10s", man[ha[i][M]]);
else
printf("%-10.10s", "-");
if (ha[i][D] >= 0)
printf("%-10.10s", drink[ha[i][D]]);
else
printf("%-10.10s", "-");
if (ha[i][A] >= 0)
printf("%-10.10s", animal[ha[i][A]]);
else
printf("%-10.10s", "-");
if (ha[i][S] >= 0)
printf("%-10.10s\n", smoke[ha[i][S]]);
else
printf("-\n");
```

```
}
}
int checkHouses(int ha[5][5]) {
int c_add = 0, c_or = 0;
int m_add = 0, m_or = 0;
int d_add = 0, d_or = 0;
int a_add = 0, a_or = 0;
int s_add = 0, s_or = 0;
// Cond 9: In the middle house they drink milk.
if (ha[2][D] >= 0 && ha[2][D] != Milk)
return Invalid;
// Cond 10: The Norwegian lives in the first house.
if (ha[0][M] >= 0 && ha[0][M] != Norwegian)
return Invalid;
for (int i = 0; i < 5; i++) {
// Uniqueness tests.
if (ha[i][C] >= 0) {
c_add += (1 << ha[i][C]);
c_or |= (1 << ha[i][C]);
}
if (ha[i][M] >= 0) {

m_add += (1 << ha[i][M]);
m_or |= (1 << ha[i][M]);
}
if (ha[i][D] >= 0) {
d_add += (1 << ha[i][D]);
d_or |= (1 << ha[i][D]);
}
if (ha[i][A] >= 0) {
a_add += (1 << ha[i][A]);
a_or |= (1 << ha[i][A]);
}
if (ha[i][S] >= 0) {
s_add += (1 << ha[i][S]);
s_or |= (1 << ha[i][S]);
}
// Cond 2: The English man lives in the red house.
if ((ha[i][M] >= 0 && ha[i][C] >= 0) &&
((ha[i][M] == English && ha[i][C] != Red) || // Checking both
(ha[i][M] != English && ha[i][C] == Red))) // to make things quicker.
return Invalid;
// Cond 3: The Swede has a dog.
if ((ha[i][M] >= 0 && ha[i][A] >= 0) &&
((ha[i][M] == Swede && ha[i][A] != Dog) ||
(ha[i][M] != Swede && ha[i][A] == Dog)))
return Invalid;
// Cond 4: The Dane drinks tea.
if ((ha[i][M] >= 0 && ha[i][D] >= 0) &&
((ha[i][M] == Dane && ha[i][D] != Tea) ||
(ha[i][M] != Dane && ha[i][D] == Tea)))
return Invalid;
// Cond 5: The green house is immediately to the left of the white house.
if ((i > 0 && ha[i][C] >= 0 /*&& ha[i-1][C] >= 0 */ ) &&
((ha[i - 1][C] == Green && ha[i][C] != White) ||
(ha[i - 1][C] != Green && ha[i][C] == White)))
return Invalid;
```

```
// Cond 6: drink coffee in the green house.
if ((ha[i][C] >= 0 && ha[i][D] >= 0) &&
((ha[i][C] == Green && ha[i][D] != Coffee) ||
(ha[i][C] != Green && ha[i][D] == Coffee)))
return Invalid;
// Cond 7: The man who smokes Pall Mall has birds.
if ((ha[i][S] >= 0 && ha[i][A] >= 0) &&

((ha[i][S] == PallMall && ha[i][A] != Birds) ||
(ha[i][S] != PallMall && ha[i][A] == Birds)))
return Invalid;
// Cond 8: In the yellow house they smoke Dunhill.
if ((ha[i][S] >= 0 && ha[i][C] >= 0) &&
((ha[i][S] == Dunhill && ha[i][C] != Yellow) ||
(ha[i][S] != Dunhill && ha[i][C] == Yellow)))
return Invalid;
// Cond 11: The man who smokes Blend lives in the house next to the house
with cats.
if (ha[i][S] == Blend) {
if (i == 0 && ha[i + 1][A] >= 0 && ha[i + 1][A] != Cats)
return Invalid;
else if (i == 4 && ha[i - 1][A] != Cats)
return Invalid;
else if (ha[i + 1][A] >= 0 && ha[i + 1][A] != Cats && ha[i - 1][A] != Cats)
return Invalid;
}
// Cond 12: In a house next to the house where they have a horse, they
smoke Dunhill.
if (ha[i][S] == Dunhill) {
if (i == 0 && ha[i + 1][A] >= 0 && ha[i + 1][A] != Horse)
return Invalid;
else if (i == 4 && ha[i - 1][A] != Horse)
return Invalid;
else if (ha[i + 1][A] >= 0 && ha[i + 1][A] != Horse && ha[i - 1][A] !=
Horse)
return Invalid;
}
// Cond 13: The man who smokes Blue Master drinks beer.
if ((ha[i][S] >= 0 && ha[i][D] >= 0) &&
((ha[i][S] == BlueMaster && ha[i][D] != Beer) ||
(ha[i][S] != BlueMaster && ha[i][D] == Beer)))
return Invalid;
// Cond 14: The German smokes Prince
if ((ha[i][M] >= 0 && ha[i][S] >= 0) &&
((ha[i][M] == German && ha[i][S] != Prince) ||
(ha[i][M] != German && ha[i][S] == Prince)))
return Invalid;
// Cond 15: The Norwegian lives next to the blue house.
if (ha[i][M] == Norwegian &&
((i < 4 && ha[i + 1][C] >= 0 && ha[i + 1][C] != Blue) ||
(i > 0 && ha[i - 1][C] != Blue)))
return Invalid;

// Cond 16: They drink water in a house next to the house where they smoke
Blend.
if (ha[i][S] == Blend) {
if (i == 0 && ha[i + 1][D] >= 0 && ha[i + 1][D] != Water)
```

```
return Invalid;
else if (i == 4 && ha[i - 1][D] != Water)
return Invalid;
else if (ha[i + 1][D] >= 0 && ha[i + 1][D] != Water && ha[i - 1][D] !=
Water)
return Invalid;
}
}
if ((c_add != c_or) || (m_add != m_or) || (d_add != d_or)
|| (a_add != a_or) || (s_add != s_or)) {
return Invalid;
}
if ((c_add != 0b11111) || (m_add != 0b11111) || (d_add != 0b11111)
|| (a_add != 0b11111) || (s_add != 0b11111)) {
return Underfull;
}
return Valid;
}

int bruteFill(int ha[5][5], int hno, int attr) {
int stat = checkHouses(ha);
if ((stat == Valid) || (stat == Invalid))
return stat;
int hb[5][5];
memcpy(hb, ha, sizeof(int) * 5 * 5);
for (int i = 0; i < 5; i++) {
hb[hno][attr] = i;
stat = checkHouses(hb);
if (stat != Invalid) {
int nexthno, nextattr;
if (attr < 4) {
nextattr = attr + 1;
nexthno = hno;
} else {
nextattr = 0;
nexthno = hno + 1;
}
stat = bruteFill(hb, nexthno, nextattr);
if (stat != Invalid) {

memcpy(ha, hb, sizeof(int) * 5 * 5);
return stat;
}
}
}
// We only come here if none of the attr values assigned were valid.
return Invalid;
}

int main() {
int ha[5][5] = {{-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}};
bruteFill(ha, 0, 0);
printHouses(ha);
return 0;
}
```

OUTPUT:

House Color Man Drink Animal Smoke
0 Yellow Norwegian Water Cats Dunhill
1 Blue Dane Tea Horse Blend
2 Red English Milk Birds PallMall
3 Green German Coffee Zebra Prince
4 White Swede Beer Dog BlueMaster

Result

The program was successfully completed & executed.

Ex No: 10 **Name:**

Date: **Reg No:**

Program for solving A* shortest path algorithm.

Aim

Program for solving A* shortest path algorithm.

PROGRAM

```
#include <limits.h>
#include <stdio.h>
#define V 9
int minDistance(int dist[], bool sptSet[]) {
int min = INT_MAX, min_index;
for (int v = 0; v < V; v++)
if (sptSet[v] == false && dist[v] <= min)
min = dist[v], min_index = v;
return min_index;
}
int printSolution(int dist[], int n) {
printf("Vertex Distance from Source\n");
for (int i = 0; i < V; i++)
printf("%d \t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src) {
int dist[V];
bool sptSet[V];
for (int i = 0; i < V; i++)
dist[i] = INT_MAX, sptSet[i] = false;
dist[src] = 0;
for (int count = 0; count < V - 1; count++) {
int u = minDistance(dist, sptSet);
sptSet[u] = true;
for (int v = 0; v < V; v++)
if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] +
graph[u][v] < dist[v]) dist[v] =
dist[u] + graph[u][v];
}
printSolution(dist, V);

}
int main() {
int graph[V][V] = { { 0, 6, 0, 0, 0, 0, 0, 8, 0 },
{ 6, 0, 8, 0, 0, 0, 0, 13, 0 },
{ 0, 8, 0, 7, 0, 6, 0, 0, 2 },
{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },
{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },
{ 0, 0, 6, 14, 10, 0, 2, 0, 0 },
{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },
{ 8, 13, 0, 0, 0, 0, 1, 0, 7 },
{ 0, 0, 2, 0, 0, 0, 6, 7, 0 }
```

```
};
dijkstra(graph, 0);
return 0;
}
```

OUTPUT

Vertex Distance from Source

0 0
1 6
2 14
3 21
4 21
5 11
6 9

Result

The program was successfully completed & executed.

Ex No: 11 **Name:**

Date: **Reg No:**

Program which demonstrates Best First Search.

Aim

Program to demonstrate Best First Search

PROGRAM

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> pi;

vector<vector<pi> > graph;

// Function for adding edges to graph
void addedge(int x, int y, int cost)
{
    graph[x].push_back(make_pair(cost, y));
    graph[y].push_back(make_pair(cost, x));
}

// Function For Implementing Best First Search
// Gives output path having lowest cost
void best_first_search(int source, int target, int n)
{
    vector<bool> visited(n, false);
    // MIN HEAP priority queue
    priority_queue<pi, vector<pi>, greater<pi> > pq;
    // sorting in pq gets done by first value of pair
    pq.push(make_pair(0, source));
    int s = source;
    visited[s] = true;
    while (!pq.empty()) {
        int x = pq.top().second;
        // Displaying the path having lowest cost
        cout << x << " ";
        pq.pop();
        if (x == target)
            break;

        for (int i = 0; i < graph[x].size(); i++) {
            if (!visited[graph[x][i].second]) {
                visited[graph[x][i].second] = true;
                pq.push(make_pair(graph[x][i].first,graph[x][i].second));
            }
        }
    }
}

// Driver code to test above methods
int main()
{
    // No. of Nodes
    int v = 14;
```

```
    graph.resize(v);

    // The nodes shown in above example(by alphabets) are
    // implemented using integers addedge(x,y,cost);
    addedge(0, 1, 3);
    addedge(0, 2, 6);
    addedge(0, 3, 5);
    addedge(1, 4, 9);
    addedge(1, 5, 8);
    addedge(2, 6, 12);
    addedge(2, 7, 14);
    addedge(3, 8, 7);
    addedge(8, 9, 5);
    addedge(8, 10, 6);
    addedge(9, 11, 1);
    addedge(9, 12, 10);
    addedge(9, 13, 2);

    int source = 0;
    int target = 9;

    // Function call
    best_first_search(source, target, v);

    return 0;
}
```

OUTPUT

0 1 3 2 8 9

Result

The program was successfully completed & executed.

Ex No: 12 **Name:**

Date: **Reg No:**

Program to solve 8-Queens problem

Aim

Program to solve 8-Queens problem.

PROGRAM

```
/* C/C++ program to solve N Queen Problem using backtracking */
#define N 8
#include <stdbool.h> #include <stdio.h>

/* A utility function to print solution */ void printSolution(int
board[N][N])
{
for (int i = 0; i < N; i++) {
for (int j = 0; j < N; j++)
printf(" %d ", board[i][j]); printf("\n");
}
}

/* A utility function to check if a queen can be placed on board[row][col].
Note that this function is called when "col" queens are already placed in
columns from 0 to col -1. So we need to check only left side for attacking
queens */
bool isSafe(int board[N][N], int row, int col)
{
int i, j;

/* Check this row on left side */ for (i = 0; i < col; i++)
if (board[row][i])
return false;
/* Check upper diagonal on left side */ for (i = row, j = col; i >= 0 && j
>= 0; i--, j--)
if (board[i][j])
return false;

/* Check lower diagonal on left side */
for (i = row, j = col; j >= 0 && i < N; i++, j--) if (board[i][j])
return false;

return true;
}

/* A recursive utility function to solve N Queen problem */
bool solveNQUtil(int board[N][N], int col)
{
/* base case: If all queens are placed then return true */
if (col >= N)
return true;

/* Consider this column and try placing this queen in all rows one by one
```

```
*/ for (int i = 0; i < N; i++) {
/* Check if the queen can be placed on board[i][col] */
if (isSafe(board, i, col)) {
/* Place this queen in board[i][col] */ board[i][col] = 1;

/* recur to place rest of the queens */ if (solveNQUtil(board, col + 1))
return true;

/* If placing queen in board[i][col] doesn't lead to a solution, then
remove queen from board[i][col] */ board[i][col] = 0; // BACKTRACK
}
}

/* If the queen cannot be placed in any row in this column col then return
false */
return false;
}
/* This function solves the N Queen problem using Backtracking. It mainly
uses solveNQUtil() to
solve the problem. It returns false if queens cannot be placed, otherwise,
return true and prints placement of queens in the form of 1s. Please note
that there may be more than one solutions, this function prints one of the
feasible solutions.*/
bool solveNQ()
{
int board[N][N] = { { 0, 0, 0, 0 },
{ 0, 0, 0, 0 },
{ 0, 0, 0, 0 },
{ 0, 0, 0, 0 } };

if (solveNQUtil(board, 0) == false) { printf("Solution does not exist");
return false;
}

printSolution(board); return true;
}

// driver program to test above function int main()
{
solveNQ(); return 0;
}
```

OUTPUT:

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1      0
0 0 0 0 1 0 0      0
0 0 0 0 0 0 0      1
0 1 0 0 0 0 0      0
0 0 0 1 0 0 0      0
0 0 0 0 0 1 0      0
0 0 1 0 0 0 0      0
```

Ex No: 13 **Name:**

Date: **Reg No:**

Program which demonstrate the precedence properties of operators in C language

Aim

Program to demonstrate the precedence properties of operators in C language.

PROGRAM

```
#include <stdio.h>

int main() {
   // arithmetic operator precedence
   int a = 10, b = 20, c = 30, result;

   result = a * b + ++c;

   printf("The result is: %d", result);

   return 0;
}
```

OUTPUT

The result is: 231

Result

The program was successfully completed & executed.

Ex No: 14 **Name:**

Date: **Reg No:**

Program to calculate factorial of a number

Aim

Program to calculate factorial of a number.

PROGRAM

```
#include <stdio.h>
int main() {
    int n, i;
    unsigned long long fact = 1;
    printf("Enter an integer: ");
    scanf("%d", &n);

    // shows error if the user enters a negative integer
    if (n < 0)
        printf("Error! Factorial of a negative number doesn't exist.");
    else {
        for (i = 1; i <= n; ++i) {
            fact *= i;
        }
        printf("Factorial of %d = %llu", n, fact);
    }

    return 0;
}
```

OUTPUT

Enter a number: 8
Factorial of 8 is: 40320

Result

The program was successfully completed & executed.

Ex No: 15 **Name:**

Date: **Reg No:**

Program to implement five House logic puzzle problem

Aim

Program to implement five House logic puzzle problem same as 9th program.

PROGRAM

```
#include <stdio.h>
#include <string.h>
enum HouseStatus { Invalid, Underfull, Valid };
enum Attrib { C, M, D, A, S };
// Unfilled attributes are represented by -1
enum Colors { Red, Green, White, Yellow, Blue };
enum Mans { English, Swede, Dane, German, Norwegian };
enum Drinks { Tea, Coffee, Milk, Beer, Water };
enum Animals { Dog, Birds, Cats, Horse, Zebra };
enum Smokes { PallMall, Dunhill, Blend, BlueMaster, Prince };
void printHouses(int ha[5][5]) {
const char *color[] = { "Red", "Green", "White", "Yellow", "Blue" };
const char *man[] = { "English", "Swede", "Dane", "German", "Norwegian" };
const char *drink[] = { "Tea", "Coffee", "Milk", "Beer", "Water" };
const char *animal[] = { "Dog", "Birds", "Cats", "Horse", "Zebra" };
const char *smoke[] = { "PallMall", "Dunhill", "Blend", "BlueMaster",
"Prince" };
printf("%-10.10s%-10.10s%-10.10s%-10.10s%-10.10s%-10.10s\n",

"House", "Color", "Man", "Drink", "Animal", "Smoke");
for (int i = 0; i < 5; i++) {
printf("%-10d", i);
if (ha[i][C] >= 0)
printf("%-10.10s", color[ha[i][C]]);
else
printf("%-10.10s", "-");
if (ha[i][M] >= 0)
printf("%-10.10s", man[ha[i][M]]);
else
printf("%-10.10s", "-");
if (ha[i][D] >= 0)
printf("%-10.10s", drink[ha[i][D]]);
else
printf("%-10.10s", "-");
if (ha[i][A] >= 0)
printf("%-10.10s", animal[ha[i][A]]);
else
printf("%-10.10s", "-");
if (ha[i][S] >= 0)
printf("%-10.10s\n", smoke[ha[i][S]]);
else
printf("-\n");
}
```

```
}
int checkHouses(int ha[5][5]) {
int c_add = 0, c_or = 0;
int m_add = 0, m_or = 0;
int d_add = 0, d_or = 0;
int a_add = 0, a_or = 0;
int s_add = 0, s_or = 0;
// Cond 9: In the middle house they drink milk.
if (ha[2][D] >= 0 && ha[2][D] != Milk)
return Invalid;
// Cond 10: The Norwegian lives in the first house.
if (ha[0][M] >= 0 && ha[0][M] != Norwegian)
return Invalid;
for (int i = 0; i < 5; i++) {
// Uniqueness tests.
if (ha[i][C] >= 0) {
c_add += (1 << ha[i][C]);
c_or |= (1 << ha[i][C]);
}
if (ha[i][M] >= 0) {

m_add += (1 << ha[i][M]);
m_or |= (1 << ha[i][M]);
}
if (ha[i][D] >= 0) {
d_add += (1 << ha[i][D]);
d_or |= (1 << ha[i][D]);
}
if (ha[i][A] >= 0) {
a_add += (1 << ha[i][A]);
a_or |= (1 << ha[i][A]);
}
if (ha[i][S] >= 0) {
s_add += (1 << ha[i][S]);
s_or |= (1 << ha[i][S]);
}
// Cond 2: The English man lives in the red house.
if ((ha[i][M] >= 0 && ha[i][C] >= 0) &&
((ha[i][M] == English && ha[i][C] != Red) || // Checking both
(ha[i][M] != English && ha[i][C] == Red))) // to make things quicker.
return Invalid;
// Cond 3: The Swede has a dog.
if ((ha[i][M] >= 0 && ha[i][A] >= 0) &&
((ha[i][M] == Swede && ha[i][A] != Dog) ||
(ha[i][M] != Swede && ha[i][A] == Dog)))
return Invalid;
// Cond 4: The Dane drinks tea.
if ((ha[i][M] >= 0 && ha[i][D] >= 0) &&
((ha[i][M] == Dane && ha[i][D] != Tea) ||
(ha[i][M] != Dane && ha[i][D] == Tea)))
return Invalid;
// Cond 5: The green house is immediately to the left of the white house.
if ((i > 0 && ha[i][C] >= 0 /*&& ha[i-1][C] >= 0 */ ) &&
((ha[i - 1][C] == Green && ha[i][C] != White) ||
(ha[i - 1][C] != Green && ha[i][C] == White)))
return Invalid;
// Cond 6: drink coffee in the green house.
```

```
if ((ha[i][C] >= 0 && ha[i][D] >= 0) &&
((ha[i][C] == Green && ha[i][D] != Coffee) ||
(ha[i][C] != Green && ha[i][D] == Coffee)))
return Invalid;
// Cond 7: The man who smokes Pall Mall has birds.
if ((ha[i][S] >= 0 && ha[i][A] >= 0) &&

((ha[i][S] == PallMall && ha[i][A] != Birds) ||
(ha[i][S] != PallMall && ha[i][A] == Birds)))
return Invalid;
// Cond 8: In the yellow house they smoke Dunhill.
if ((ha[i][S] >= 0 && ha[i][C] >= 0) &&
((ha[i][S] == Dunhill && ha[i][C] != Yellow) ||
(ha[i][S] != Dunhill && ha[i][C] == Yellow)))
return Invalid;
// Cond 11: The man who smokes Blend lives in the house next to the house
with cats.
if (ha[i][S] == Blend) {
if (i == 0 && ha[i + 1][A] >= 0 && ha[i + 1][A] != Cats)
return Invalid;
else if (i == 4 && ha[i - 1][A] != Cats)
return Invalid;
else if (ha[i + 1][A] >= 0 && ha[i + 1][A] != Cats && ha[i - 1][A] != Cats)
return Invalid;
}
// Cond 12: In a house next to the house where they have a horse, they
smoke Dunhill.
if (ha[i][S] == Dunhill) {
if (i == 0 && ha[i + 1][A] >= 0 && ha[i + 1][A] != Horse)
return Invalid;
else if (i == 4 && ha[i - 1][A] != Horse)
return Invalid;
else if (ha[i + 1][A] >= 0 && ha[i + 1][A] != Horse && ha[i - 1][A] !=
Horse)
return Invalid;
}
// Cond 13: The man who smokes Blue Master drinks beer.
if ((ha[i][S] >= 0 && ha[i][D] >= 0) &&
((ha[i][S] == BlueMaster && ha[i][D] != Beer) ||
(ha[i][S] != BlueMaster && ha[i][D] == Beer)))
return Invalid;
// Cond 14: The German smokes Prince
if ((ha[i][M] >= 0 && ha[i][S] >= 0) &&
((ha[i][M] == German && ha[i][S] != Prince) ||
(ha[i][M] != German && ha[i][S] == Prince)))
return Invalid;
// Cond 15: The Norwegian lives next to the blue house.
if (ha[i][M] == Norwegian &&
((i < 4 && ha[i + 1][C] >= 0 && ha[i + 1][C] != Blue) ||
(i > 0 && ha[i - 1][C] != Blue)))
return Invalid;

// Cond 16: They drink water in a house next to the house where they smoke
Blend.
if (ha[i][S] == Blend) {
if (i == 0 && ha[i + 1][D] >= 0 && ha[i + 1][D] != Water)
return Invalid;
```

```
else if (i == 4 && ha[i - 1][D] != Water)
return Invalid;
else if (ha[i + 1][D] >= 0 && ha[i + 1][D] != Water && ha[i - 1][D] !=
Water)
return Invalid;
}
}
if ((c_add != c_or) || (m_add != m_or) || (d_add != d_or)
|| (a_add != a_or) || (s_add != s_or)) {
return Invalid;
}
if ((c_add != 0b11111) || (m_add != 0b11111) || (d_add != 0b11111)
|| (a_add != 0b11111) || (s_add != 0b11111)) {
return Underfull;
}
return Valid;
}

int bruteFill(int ha[5][5], int hno, int attr) {
int stat = checkHouses(ha);
if ((stat == Valid) || (stat == Invalid))
return stat;
int hb[5][5];
memcpy(hb, ha, sizeof(int) * 5 * 5);
for (int i = 0; i < 5; i++) {
hb[hno][attr] = i;
stat = checkHouses(hb);
if (stat != Invalid) {
int nexthno, nextattr;
if (attr < 4) {
nextattr = attr + 1;
nexthno = hno;
} else {
nextattr = 0;
nexthno = hno + 1;
}
stat = bruteFill(hb, nexthno, nextattr);
if (stat != Invalid) {

memcpy(ha, hb, sizeof(int) * 5 * 5);
return stat;
}
}
}
// We only come here if none of the attr values assigned were valid.
return Invalid;
}

int main() {
int ha[5][5] = {{-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}};
bruteFill(ha, 0, 0);
printHouses(ha);
return 0;
}
```

OUTPUT:

House Color Man Drink Animal Smoke
0 Yellow Norwegian Water Cats Dunhill
1 Blue Dane Tea Horse Blend
2 Red English Milk Birds PallMall
3 Green German Coffee Zebra Prince
4 White Swede Beer Dog BlueMaster

Result

The program was successfully completed & executed.