/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Welcome to GDB Online.

GDB online is an online compiler and debugger tool for C, C++, Python, PHP, Ruby,

C#, VB, Perl, Swift, Prolog, Javascript, Pascal, HTML, CSS, JS

Code, Compile, Run and Debug online from anywhere in world.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

// Maze size

#define N 4

#define bool int

#define true 1

#define false 0

int traversed[N][N] = { {1,1,1,1,1},

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

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

{1,1,1,1,1}

};

bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N]);

/\* A utility function to print solution matrix sol[N][N] \*/

void printSolution(int sol[N][N])

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

printf(" %d ", sol[i][j]);

printf("\n");

}

}

/\* A utility function to check if x,y is valid index for N\*N maze \*/

bool isSafe(int maze[N][N], int x, int y)

{

// if (x,y outside maze) return false

if(x >= 0 && x < N && y >= 0 && y < N && maze[x][y] &&traversed[x][y]== 1)

return true;

return false;

}

/\* This function solves the Maze problem using Backtracking. It mainly

uses solveMazeUtil() to solve the problem. It returns false if no

path is possible, otherwise return true and prints the path in the

form of 1s. Please note that there may be more than one solutions,

this function prints one of the feasible solutions.\*/

bool solveMaze(int maze[N][N])

{

int sol[N][N] = { {0, 0, 0, 0,0},

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

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

{0, 0, 0, 0,0}

};

if(solveMazeUtil(maze, 0, 0, sol) == false)

{

printf("Solution doesn't exist");

return false;

}

printSolution(sol);

return true;

}

/\* A recursive utility function to solve Maze problem \*/

bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N])

{

// if (x,y is goal) return true

if(x == N-1 && y == N-1)

{

sol[x][y] = 1;

return true;

}

// Check if maze[x][y] is valid

if(isSafe(maze, x, y) == true)

{

// mark x,y as part of solution path

sol[x][y] = 1;

traversed[x][y]=0;

/\* Move forward in x direction \*/

if (solveMazeUtil(maze, x+1, y, sol) == true)

return true;

/\* If moving in x direction doesn't give solution then

Move down in y direction \*/

if (solveMazeUtil(maze, x, y+1, sol) == true)

return true;

if (solveMazeUtil(maze, x-1, y, sol) == true)

return true;

/\* If moving in x direction doesn't give solution then

Move down in y direction \*/

if (solveMazeUtil(maze, x, y-1, sol) == true)

return true;

/\* If none of the above movements work then BACKTRACK:

unmark x,y as part of solution path \*/

sol[x][y] = 0;

return false;

}

return false;

}

// driver program to test above function

int main()

{

int maze[N][N] =

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

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

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

{1, 0, 1, 1,1}

};

solveMaze(maze);

return 0;

}