**ASSIGNMENT - 1**

**Transposition Cipher**

// Java Program to Perform Cryptography

// using Transposition Technique

// Importing all classes from

// java.util package

// Importing input output classes

import java.io.\*;

import java.util.\*;

// Class

// For transposition cipher

public class GFG {

// Member variables of this class

public static String selectedKey;

public static char sortedKey[];

public static int sortedKeyPos[];

// Constructor 1 of this class

// Default constructor defining the default key

public GFG()

{

selectedKey = "megabuck";

sortedKeyPos = new int[selectedKey.length()];

sortedKey = selectedKey.toCharArray();

}

// Constructor 2 of this class

// Parameterized constructor defining the custom key

public GFG(String GeeksForGeeks)

{

selectedKey = GeeksForGeeks;

sortedKeyPos = new int[selectedKey.length()];

sortedKey = selectedKey.toCharArray();

}

// Method 1 - doProcessOnKey()

// To reorder data do the sorting on selected key

public static void doProcessOnKey()

{

// Find position of each character in selected key

// and arranging it in alphabetical order

int min, i, j;

char orginalKey[] = selectedKey.toCharArray();

char temp;

// Step 1: Sorting the array of selected key

// using nested for loops

for (i = 0; i < selectedKey.length(); i++) {

min = i;

for (j = i; j < selectedKey.length(); j++) {

if (sortedKey[min] > sortedKey[j]) {

min = j;

}

}

if (min != i) {

temp = sortedKey[i];

sortedKey[i] = sortedKey[min];

sortedKey[min] = temp;

}

}

// Step 2: Filling the position of array

// according to alphabetical order

// using nested for loops

for (i = 0; i < selectedKey.length(); i++) {

for (j = 0; j < selectedKey.length(); j++) {

if (orginalKey[i] == sortedKey[j])

sortedKeyPos[i] = j;

}

}

}

// Method 2 - doEncryption()

// To encrypt the targeted string

public static String doEncryption(String plainText)

{

int min, i, j;

char orginalKey[] = selectedKey.toCharArray();

char temp;

doProcessOnKey();

// Step 3: Generating the encrypted message by

// doing encryption using Transpotion Cipher

int row = plainText.length() / selectedKey.length();

int extrabit

= plainText.length() % selectedKey.length();

int exrow = (extrabit == 0) ? 0 : 1;

int rowtemp = -1, coltemp = -1;

int totallen = (row + exrow) \* selectedKey.length();

char pmat[][] = new char[(row + exrow)]

[(selectedKey.length())];

char encry[] = new char[totallen];

int tempcnt = -1;

row = 0;

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

coltemp++;

if (i < plainText.length()) {

if (coltemp == (selectedKey.length())) {

row++;

coltemp = 0;

}

pmat[row][coltemp] = plainText.charAt(i);

}

else {

// Padding can be added between two

// consecutive alphabets or a group of

// alphabets of the resultant cipher text

pmat[row][coltemp] = '-';

}

}

int len = -1, k;

for (i = 0; i < selectedKey.length(); i++) {

for (k = 0; k < selectedKey.length(); k++) {

if (i == sortedKeyPos[k]) {

break;

}

}

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

len++;

encry[len] = pmat[j][k];

}

}

String p1 = new String(encry);

return (new String(p1));

}

// Method 3 - doEncryption()

// To decrypt the targeted string

public static String doDecryption(String s)

{

int min, i, j, k;

char key[] = selectedKey.toCharArray();

char encry[] = s.toCharArray();

char temp;

doProcessOnKey();

// Step 4: Generating a plain message

int row = s.length();

selectedKey.length();

char pmat[][]

= new char[row][(selectedKey.length())];

int tempcnt = -1;

for (i = 0; i < selectedKey.length(); i++) {

for (k = 0; k < selectedKey.length(); k++) {

if (i == sortedKeyPos[k]) {

break;

}

}

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

tempcnt++;

pmat[j][k] = encry[tempcnt];

}

}

// Step 5: Storing matrix character in

// to a single string

char p1[] = new char[row \* selectedKey.length()];

k = 0;

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

for (j = 0; j < selectedKey.length(); j++) {

if (pmat[i][j] != '\*') {

p1[k++] = pmat[i][j];

}

}

}

p1[k++] = '\0';

return (new String(p1));

}

@SuppressWarnings("static-access")

// Method 4 - main()

// Main driver method

public static void main(String[] args)

{

// Creating object of class in main method

GFG tc = new GFG();

System.out.println("Enter Plain Text to encrypt");

Scanner sc= new Scanner(System.in);

String pt=sc.next();

// Printing the ciphere text

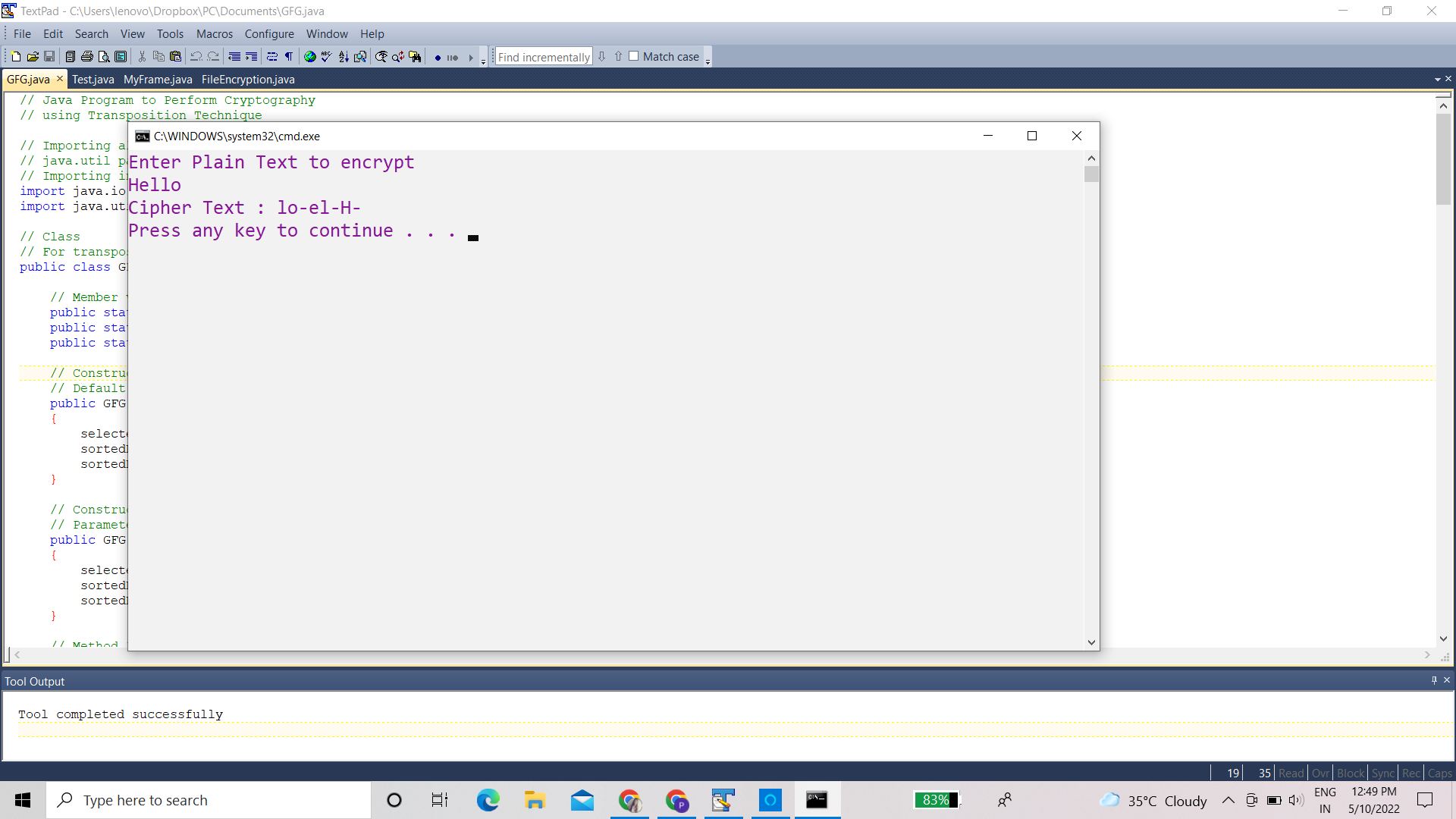
// Custom input - Hello

System.out.println("Cipher Text : "

+ tc.doEncryption(pt));

}

}



**//Transposition Cipher Short Code:**

import java.util.\*;

import java.util.Scanner; // needed for Scanner

public class Transposition {

public static void main(String args[]) {

String key;

String message;

String encryptedMessage;

// Letters in the x-axis

int x = 0;

// Letters in the y-axis

int y = 0;

// Prompt the user

System.out.print("Type your Key : ");

// Read a line of text from the user.

Scanner scan = new Scanner(System.in);

key = scan.nextLine();

// Display the input back to the user.

System.out.println("Your Key is " + key);

//Prompt the user

System.out.print("Type your Message : ");

//Read a line of text from the user.

message = scan.nextLine();

//Display the input back to the user.

System.out.println("Your Message is " + message);

int msgchar = message.length();

int keycahr = key.length();

if (!((msgchar % keycahr) == 0)) {

do {

message = message + "x";

msgchar = message.length();

} while (!((msgchar % keycahr) == 0));

}

encryptedMessage = "";

// To set the temp as [x][y]

char temp[][] = new char[key.length()][message.length()];

char msg[] = message.toCharArray();

// To populate the array

x = 0;

y = 0;

// To convert the message into an array of char

for (int i = 0; i < msg.length; i++) {

temp[x][y] = msg[i];

if (x == (key.length() - 1)) {

x = 0;

y = y + 1;

} // Close if

else {

x++;

}

} // Close for loop

// To sort the key

char t[] = new char[key.length()];

t = key.toCharArray();

Arrays.sort(t);

for (int j = 0; j < y; j++) {

for (int i = 0; i < key.length(); i++) {

System.out.print(temp[i][j]);

}

System.out.println();

}

System.out.println();

// To print out row by row (i.e. y)

for (int j = 0; j < y; j++) {

// To compare the the sorted Key with the key

// For char in the key

for (int i = 0; i < key.length(); i++) {

int pos = 0;

// To get the position of key.charAt(i) from sorted key

for (pos = 0; pos < t.length; pos++) {

if (key.charAt(i) == t[pos]) {

// To break the for loop once the key is found

break;

}

}

System.out.print(temp[pos][j]);

encryptedMessage += temp[pos][j];

}

System.out.println();

}

System.out.println(encryptedMessage);

System.exit(0);

}

}

**ASSIGNMENT - 2**

DES Assignment:-

//Java Programme

package descrypt;

import java.util.\*;

import javax.crypto.\*;

import java.io.\*;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.security.spec.InvalidKeySpecException;

public class descrypt {

public static void main(String args[]) throws IOException, NoSuchAlgorithmException, InvalidKeyException, InvalidKeySpecException, NoSuchPaddingException, IllegalBlockSizeException, BadPaddingException {

try {

BufferedReader bfn=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the String: ");

String str=bfn.readLine();

byte[] msg=str.getBytes();

KeyGenerator Mygenerator=KeyGenerator.getInstance("DES");

SecretKey mydeskey=Mygenerator.generateKey();

Cipher myCipher = Cipher.getInstance("DES");

myCipher.init(Cipher.ENCRYPT\_MODE, mydeskey);

byte[] myEncryptedBytes=myCipher.doFinal(msg);

myCipher.init(Cipher.DECRYPT\_MODE, mydeskey);

byte[] myDecryptedBytes=myCipher.doFinal(myEncryptedBytes);

String encrypteddata=new String(myEncryptedBytes);

String decrypteddata=new String(myDecryptedBytes);

System.out.println("Encrypted - "+ encrypteddata);

System.out.println("Decrypted Message - "+ decrypteddata);

}

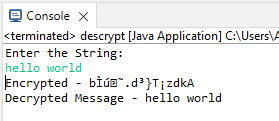
catch(Exception e) {

System.out.println("Exception caught!!!");

}

}}

**Output**



**ASSIGNMENT - 3**

**AES Algorithm**

import javax.crypto.\*;

import javax.crypto.spec.IvParameterSpec;

import java.security.InvalidAlgorithmParameterException;

import java.security.InvalidKeyException;

import java.security.NoSuchAlgorithmException;

import java.util.Base64;

import java.io.\*;/

public class AESCrypt{

public static void main(String args[]) throws NoSuchAlgorithmException,NoSuchPaddingException,

InvalidKeyException, IllegalBlockSizeException, BadPaddingException, InvalidAlgorithmParameterException,IOException {

//KeyGeneration

KeyGenerator keygenerator=KeyGenerator.getInstance("AES");

SecretKey secretkey=keygenerator.generateKey();

String secretKeyString=Base64.getEncoder().encodeToString(secretkey.getEncoded());

System.out.println("generated key: "+secretKeyString);

//Encryption of message

Cipher encryptionCipher=Cipher.getInstance("AES/CBC/PKCS5Padding");

byte[] InitVectorBytes=keygenerator.generateKey().getEncoded();

IvParameterSpec parameterSpec=new IvParameterSpec(InitVectorBytes);

encryptionCipher.init(Cipher.ENCRYPT\_MODE,secretkey,parameterSpec);

BufferedReader bfn=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the String: ");

String str=bfn.readLine();

byte[] encryptedMessageBytes=encryptionCipher.doFinal(str.getBytes());

String encryptedMessage=Base64.getEncoder().encodeToString(encryptedMessageBytes);

System.out.println("Encrypted Message: "+encryptedMessage);

//Decryption of message

Cipher decryptioncipher=Cipher.getInstance("AES/CBC/PKCS5Padding");

decryptioncipher.init(Cipher.DECRYPT\_MODE,secretkey,parameterSpec);

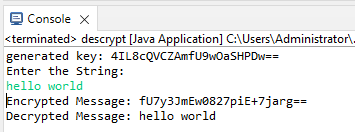
byte[] decryptedMessageBytes=decryptioncipher.doFinal(encryptedMessageBytes);

String decryptedMessage=new String(decryptedMessageBytes);

System.out.println("Decrypted Message: "+decryptedMessage);

}

**Output**

}

**ASSIGNMENT - 4**

RSA

// Java Program to Implement the RSA Algorithm

import java.io.BufferedReader; import

java.io.InputStreamReader; import

java.math.\*;

import java.util.Random;

import java.util.Scanner;

public class RSA{

static Scanner sc = new Scanner(System.in); public

static void main(String[] args) {

// TODO code application logic here

System.out.print("Enter a Prime number:");

BigInteger p = sc.nextBigInteger(); // Here's one prime number..

System.out.print("Enter another prime number:"); BigInteger q =

sc.nextBigInteger(); // ..andanother.

BigInteger n = p.multiply(q);

BigInteger n2 = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE)); BigInteger e

= generateE(n2);

BigInteger d = e.modInverse(n2); // Here's the multiplicative inverse

System.out.println("Encryption keys are: " + e + ", " + n);

System.out.println("Decryption keys are: " + d + ", " + n);

}

public static BigIntegergenerateE(BigIntegerfiofn) { int y,

intGCD;

BigInteger e;

BigInteger gcd;

Random x = new Random();

do {

y = x.nextInt(fiofn.intValue()-1);

String z = Integer.toString(y);

e = new BigInteger(z);

gcd = fiofn.gcd(e);

intGCD = gcd.intValue();

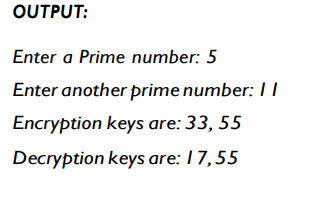
}

while(y <= 2 || intGCD != 1);

return e;

}

}



**ASSIGNMENT - 5**

**MD5**

import java.security.\*;

import java.math.\*;

import java.util.Scanner;

/\*\*

\*

\* @author ADMIN

\*/

public class MD5 {

public static String MD5(String s) throws Exception {

MessageDigest m=MessageDigest.getInstance("MD5");

m.update(s.getBytes(),0,s.length());

return new BigInteger(1,m.digest()).toString(16);

}

public static void main(String args[]) throws Exception {

//Encode

System.out.println("Enter the text: ");

Scanner sc=new Scanner(System.in);

String md5=sc.next();

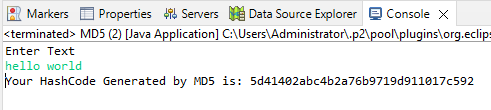
md5 = MD5("MD5Online");

System.out.println("MD5 hash: "+md5);

}

}

**Output**



**All AI assignments**

**1] BFS –**

#include &lt;iostream&gt;

#include &lt;queue&gt;

#include &lt;vector&gt;

using namespace std;

// Data structure to store a graph edge

struct Edge {

int src, dest;

};

// A class to represent a graph object

class Graph

{

public:

// a vector of vectors to represent an adjacency list

vector&lt;vector&lt;int&gt;&gt; adjList;

// Graph Constructor

Graph(vector&lt;Edge&gt; const &amp;edges, int n)

{

// resize the vector to hold `n` elements of type `vector&lt;int&gt;`

adjList.resize(n);

// add edges to the undirected graph

for (auto &amp;edge: edges)

{

adjList[edge.src].push\_back(edge.dest);

adjList[edge.dest].push\_back(edge.src);

}

}

};

// Perform BFS recursively on the graph

void recursiveBFS(Graph const &amp;graph, queue&lt;int&gt; &amp;q, vector&lt;bool&gt; &amp;discovered)

{

if (q.empty()) {

return;

}

// dequeue front node and print it

int v = q.front();

q.pop();

cout &lt;&lt; v &lt;&lt; &quot; &quot;;

// do for every edge (v, u)

for (int u: graph.adjList[v])

{

if (!discovered[u])

{

// mark it as discovered and enqueue it

discovered[u] = true;

q.push(u);

}

}

recursiveBFS(graph, q, discovered);

}

int main()

{

// vector of graph edges as per the above diagram

vector&lt;Edge&gt; edges = {

{1, 2}, {1, 3}, {1, 4}, {2, 5}, {2, 6}, {5, 9},

{5, 10}, {4, 7}, {4, 8}, {7, 11}, {7, 12}

// vertex 0, 13, and 14 are single nodes

};

// total number of nodes in the graph (labelled from 0 to 14)

int n = 15;

// build a graph from the given edges

Graph graph(edges, n);

// to keep track of whether a vertex is discovered or not

vector&lt;bool&gt; discovered(n, false);

// create a queue for doing BFS

queue&lt;int&gt; q;

// Perform BFS traversal from all undiscovered nodes to

// cover all connected components of a graph

for (int i = 0; i &lt; n; i++)

{

if (discovered[i] == false)

{

// mark the source vertex as discovered

discovered[i] = true;

// enqueue source vertex

q.push(i);

// start BFS traversal from vertex `i`

recursiveBFS(graph, q, discovered);

}

}

return 0;

}

**2] DFS –**

#include &lt;iostream&gt;

#include &lt;vector&gt;

using namespace std;

// Data structure to store a graph edge

struct Edge {

int src, dest;

};

// A class to represent a graph object

class Graph

{

public:

// a vector of vectors to represent an adjacency list

vector&lt;vector&lt;int&gt;&gt; adjList;

// Graph Constructor

Graph(vector&lt;Edge&gt; const &amp;edges, int n)

{

// resize the vector to hold `n` elements of type `vector&lt;int&gt;`

adjList.resize(n);

// add edges to the undirected graph

for (auto &amp;edge: edges)

{

adjList[edge.src].push\_back(edge.dest);

adjList[edge.dest].push\_back(edge.src);

}

}

};

// Function to perform DFS traversal on the graph on a graph

void DFS(Graph const &amp;graph, int v, vector&lt;bool&gt; &amp;discovered)

{

// mark the current node as discovered

discovered[v] = true;

// print the current node

cout &lt;&lt; v &lt;&lt; &quot; &quot;;

// do for every edge (v, u)

for (int u: graph.adjList[v])

{

// if `u` is not yet discovered

if (!discovered[u]) {

DFS(graph, u, discovered);

}

}

}

int main()

{

// vector of graph edges as per the above diagram

vector&lt;Edge&gt; edges = {

// Notice that node 0 is unconnected

{1, 2}, {1, 7}, {1, 8}, {2, 3}, {2, 6}, {3, 4},

{3, 5}, {8, 9}, {8, 12}, {9, 10}, {9, 11}

};

// total number of nodes in the graph (labelled from 0 to 12)

int n = 13;

// build a graph from the given edges

Graph graph(edges, n);

// to keep track of whether a vertex is discovered or not

vector&lt;bool&gt; discovered(n);

// Perform DFS traversal from all undiscovered nodes to

// cover all connected components of a graph

for (int i = 0; i &lt; n; i++)

{

if (discovered[i] == false) {

DFS(graph, i, discovered);

}

}

return 0;

}

**Greedy Search: Selection sort Code -**

public class SelectionSort

{

public static void main(String[] args)

{

int [] arr= {5,8,10,4,1};

for(int i = 0;i&lt;=4;++i)

{

int min=arr[i];

for(int j=i+1;j&lt;arr.length;++j)

{

if(arr[i] &gt; arr[j] )

{

min = arr[j];

arr[j] = arr[i];

arr[i] = min;

}

}

System.out.print(arr[i]+&quot; &quot;);

}

}

}

**C/C++ program to solve N Queen Problem using backtracking -**

#define N 4

#include &lt;stdbool.h&gt;

#include &lt;stdio.h&gt;

/\* A utility function to print solution \*/

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

{

for (int i = 0; i &lt; N; i++) {

for (int j = 0; j &lt; N; j++)

printf(&quot; %d &quot;, board[i][j]);

printf(&quot;\n&quot;);

}

}

/\* A utility function to check if a queen canbe placed on board[row][col]. Note

that this function is called when &quot;col&quot; queens arealready 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 &lt; col; i++)

if (board[row][i])

return false;

/\* Check upper diagonal on left side \*/

for (i = row, j = col; i &gt;= 0 &amp;&amp; j &gt;= 0; i--, j--)

if (board[i][j])

return false;

/\* Check lower diagonal on left side \*/

for (i = row, j = col; j &gt;= 0 &amp;&amp; i &lt; 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 &gt;= N)

return true;

/\* Consider this column and try placing

this queen in all rows one by one \*/

for (int i = 0; i &lt; 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&#39;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(&quot;Solution does not exist&quot;);

return false;

}

printSolution(board);

return true;

}

// driver program to test above function

int main()

{

solveNQ();

return 0;

}