**Manav Bhanushali**

**CSE-DS D1**

**2021700008**

**DAA Exp :- 7**

**AIM: Experiment based on backtracking strategy: N-Queens problem**

Algorithm:

If two queens are placed at position (i, j) and (k, l).

Then they are on same diagonal only if (i - j) = k - l or i + j = k + l.

The first equation implies that j - l = i - k.

The second equation implies that j - l = k - i.

Therefore, two queens lie on the duplicate diagonal if and only if |j-l|=|i-k|

Place (k, i) returns a Boolean value that is true if the kth queen can be placed in column i.

x[] is a global array whose final k - 1 values have been set. Abs (r) returns the absolute value

of r.

Place (k, i)

{

For j ← 1 to k - 1

do if (x [j] = i) or (Abs (x [j]) – i)) == (Abs (j - k))

then return false;

return true;

}

N - Queens (k, n):

{

For i ← 1 to n

do if Place (k, i) then

{

x [k] ← i;

if (k ==n) then

write (x [1....n));

else

N - Queens (k + 1, n);

}

}

Code :-

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <math.h>

int \*

deepCopy (int \*arr, int len)

{

int \*newarr = malloc (sizeof (int) \* len);

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

newarr[i] = arr[i];

return newarr;

}

int \*

printArray (int \*arr, int len)

{

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

printf ("%8d", arr[i]);

printf ("\n");

}

typedef struct node

{

int \*data;

struct node \*next;

} \*pnode;

typedef struct list

{

pnode head;

pnode tail;

int n;

} \*plist;

void

addNodetoList (plist l, int \*data)

{

if (l->head != NULL)

{

l->tail->next = malloc (sizeof (struct node));

l->tail = l->tail->next;

l->tail->next = NULL;

l->tail->data = data;

}

else

{

l->head = malloc (sizeof (struct node));

l->tail = l->head;

l->tail->next = NULL;

l->tail->data = data;

}

}

plist createList (int n)

{

plist l = malloc (sizeof (struct list));

l->head = NULL;

l->tail = NULL;

l->n = n;

return l;

}

typedef struct NQueensSolutions

{

int count;

plist solutions; // linked list of solutions

} \*pNQueensSolutions;

void

printSolutions (pNQueensSolutions sol)

{

int n = sol->solutions->n;

printf ("\n");

for (pnode temp = sol->solutions->head; temp != NULL; temp =

temp->next)

printArray (temp->data, n);

}

bool queenCanBePlaced (int n, int k, int pos, int \*curr\_board)

{

int a, b;

for (int i = 1; i < k; i++)

{

a = abs (i - k); // x1 - x2

b = abs (curr\_board[i - 1] - pos); // y1 -y2

if (a == b || a == 0 || b == 0)

return false;

}

return true;

}

void

placeKthQueen (int k, int n, int \*curr\_board, pNQueensSolutions

solutions)

{

for (int i = 1; i <= n; i++)

{

if (queenCanBePlaced (n, k, i, curr\_board))

{

curr\_board[k - 1] = i;

if (k == n) // this is a complete solution

{

addNodetoList (solutions->solutions,

deepCopy (curr\_board, n)); // add a deep copy of current board to the list of solutions

solutions->count++;

}

else // place the next queen

placeKthQueen (k + 1, n, curr\_board, solutions);

}

}

}

pNQueensSolutions NQueens (int n)

{

pNQueensSolutions nqs = malloc (sizeof (struct NQueensSolutions));

nqs->count = 0;

nqs->solutions = createList (n);

int curr\_board[n];

curr\_board[0] = 1;

placeKthQueen (1, n, curr\_board, nqs);

return nqs;

}

int

main ()

{

int n;

printf ("Enter number of queens: ");

scanf ("%d", &n);

pNQueensSolutions sol = NQueens (n);

printf ("\nNumber of solutions: %d\n", sol->count);

if (sol->count > 0)

{

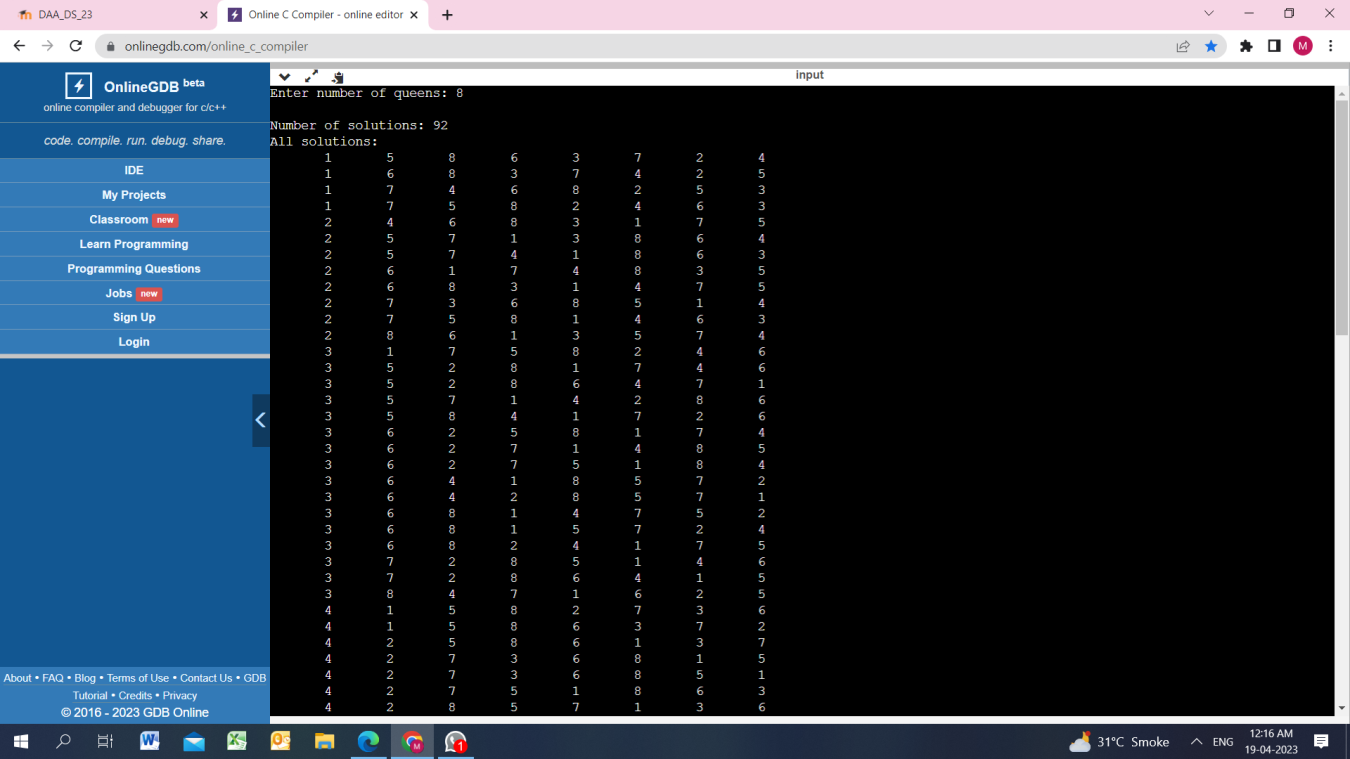
printf ("All solutions:");

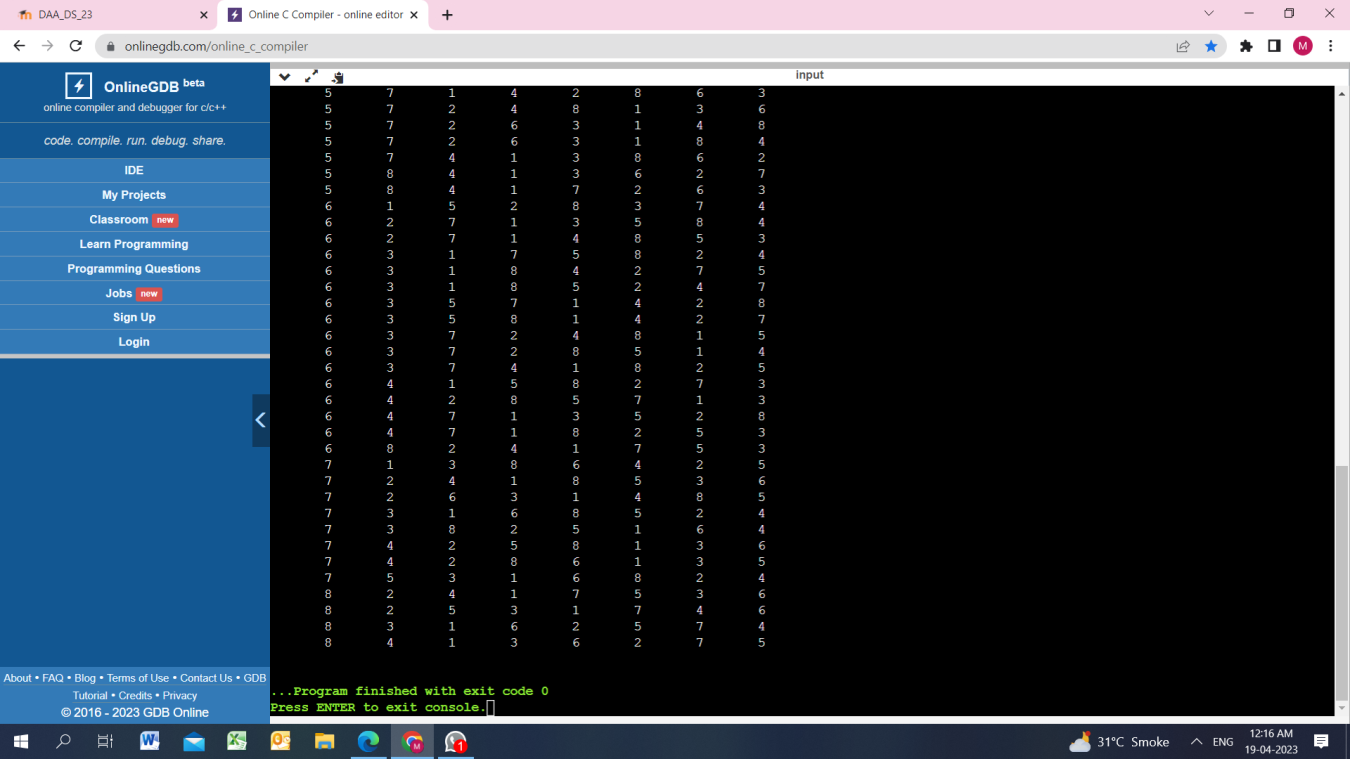
printSolutions (sol);

}

}

OUTPUT:





.