CHAPTER - 23

BACKTRACKING

CHAPTER 23

BACKTRACING

BACKTRACKING **(Reading)**

EIGHT QUEENS PROBLEM

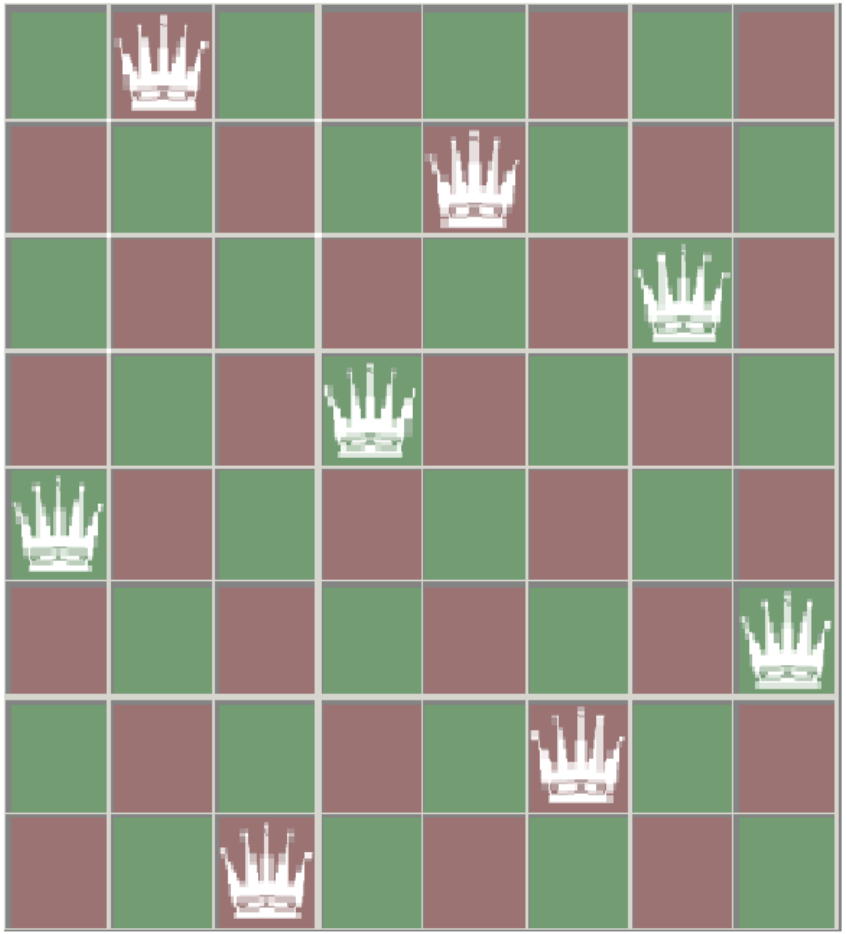
N QUEENS PROBLEM SOLUTION

BACKTRACKING SITUATION

EIGHT QUEENS SOLUTION PROGRAM

BACKTRACKING

EIGHT QUEENS PROBLEM



- There are *92* solutions to the Queens problem, there are *12* distinct patterns.
- All of the *92* solutions can be transformed into one of these *12* unique patterns using rotations and reflections to place eight queens on an *8 x 8 chessboard*.

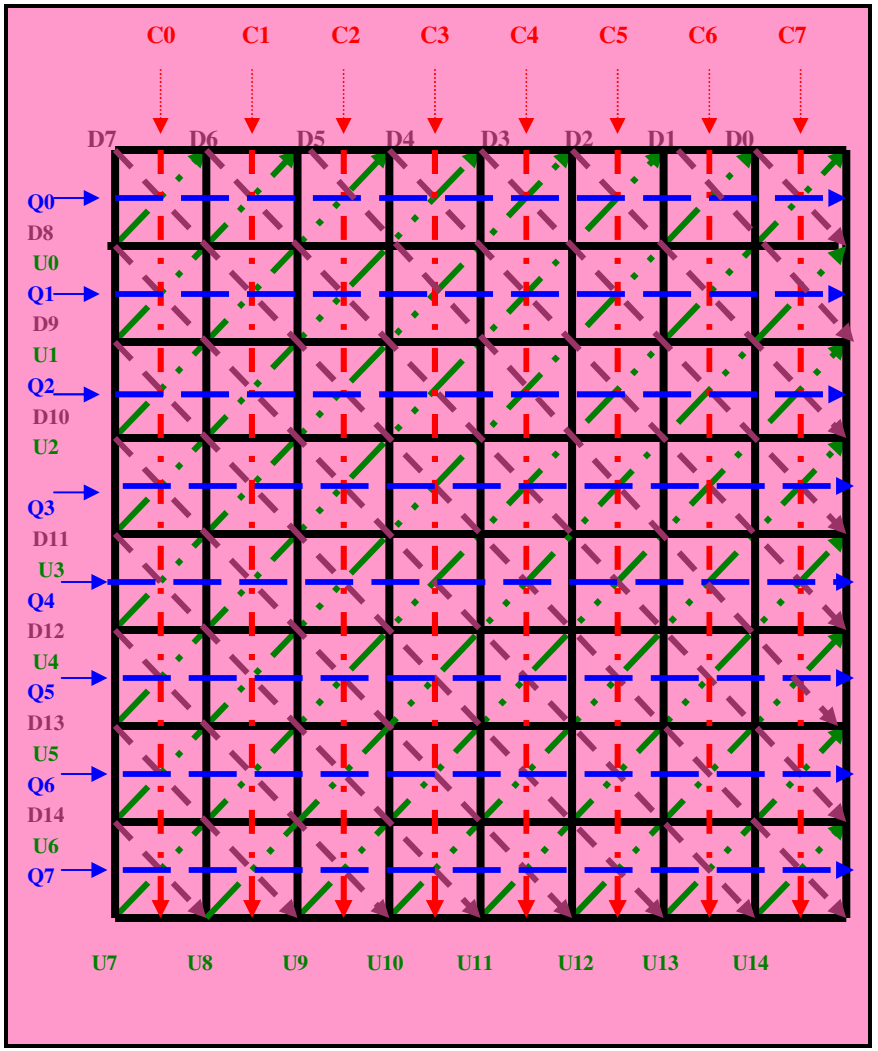
Solution Number	Row1 col	Row2 col	Row3 col	Row4 col	Row5 col	Row6 col	Row7 col	Row8 col
1	1	5	8	6	3	7	2	4
2	1	6	8	3	7	4	2	5
3	2	4	6	8	3	1	7	5
4	2	5	7	1	3	8	6	4
5	2	5	7	4	1	8	6	3
6	2	6	1	7	4	8	3	5
7	2	6	8	3	1	4	7	5
8	2	7	3	6	8	5	1	4
9	2	7	5	8	1	4	6	3
10	3	5	2	8	1	7	4	6
11	3	5	8	4	1	7	2	6
12	3	6	2	5	8	1	7	4

BACKTRACKING

N QUEENS PROBLEM SOLUTION

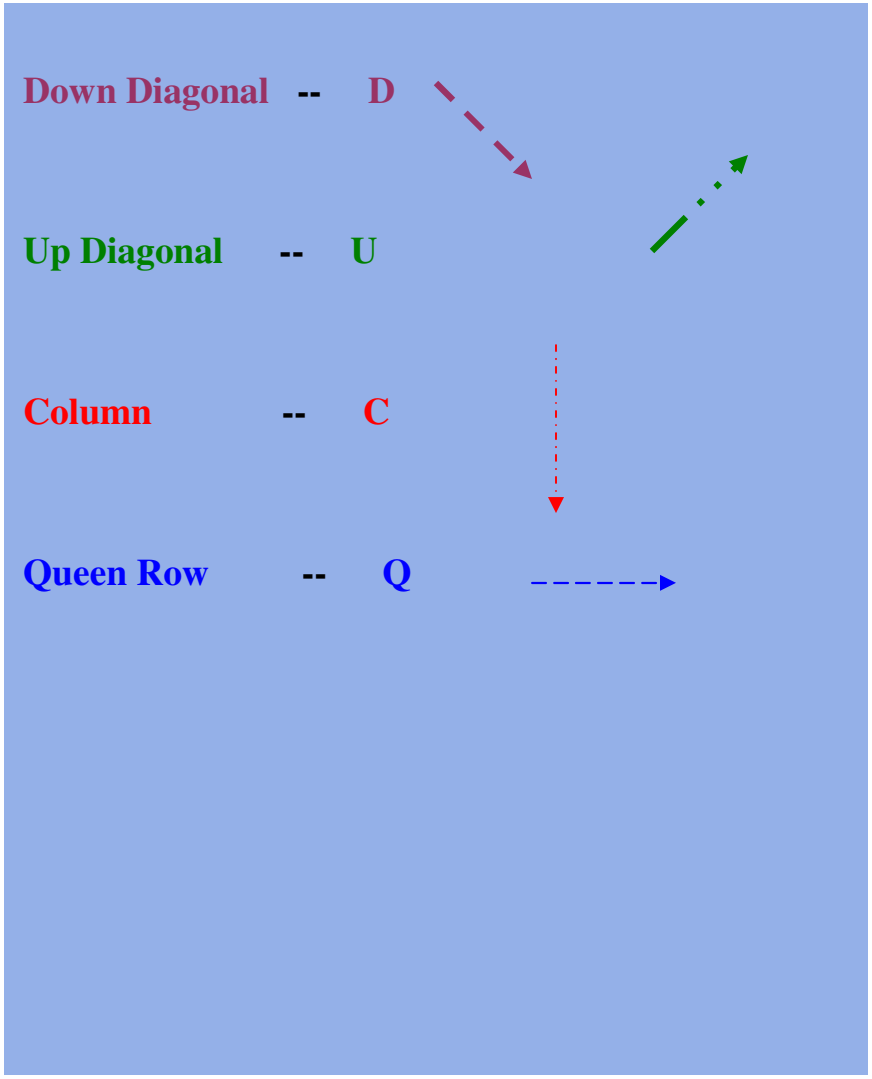
- On an *8* x *8* chessboard *8* queens are placed in such a way that no queen attacks another one.

```
void addQueen (void)
{
    for (every unguarded position p on the board)
    {
        place a queen in position p;
        n++;
        if (n == 8)
            print the configuration;
        else
            addQueen ();
        remove the queen from position p;
        n—
    }
}
```



BACKTRACKING

N QUEENS PROBLEM SOLUTION



- A **queen** is placed in a column of a row

  When there is no queen is placed in that column and Up diagonal is free of **queens** and **Down diagonal is free of queens.**
- When a **queen** is placed in a column ***n*** of a row the following initialization takes place:

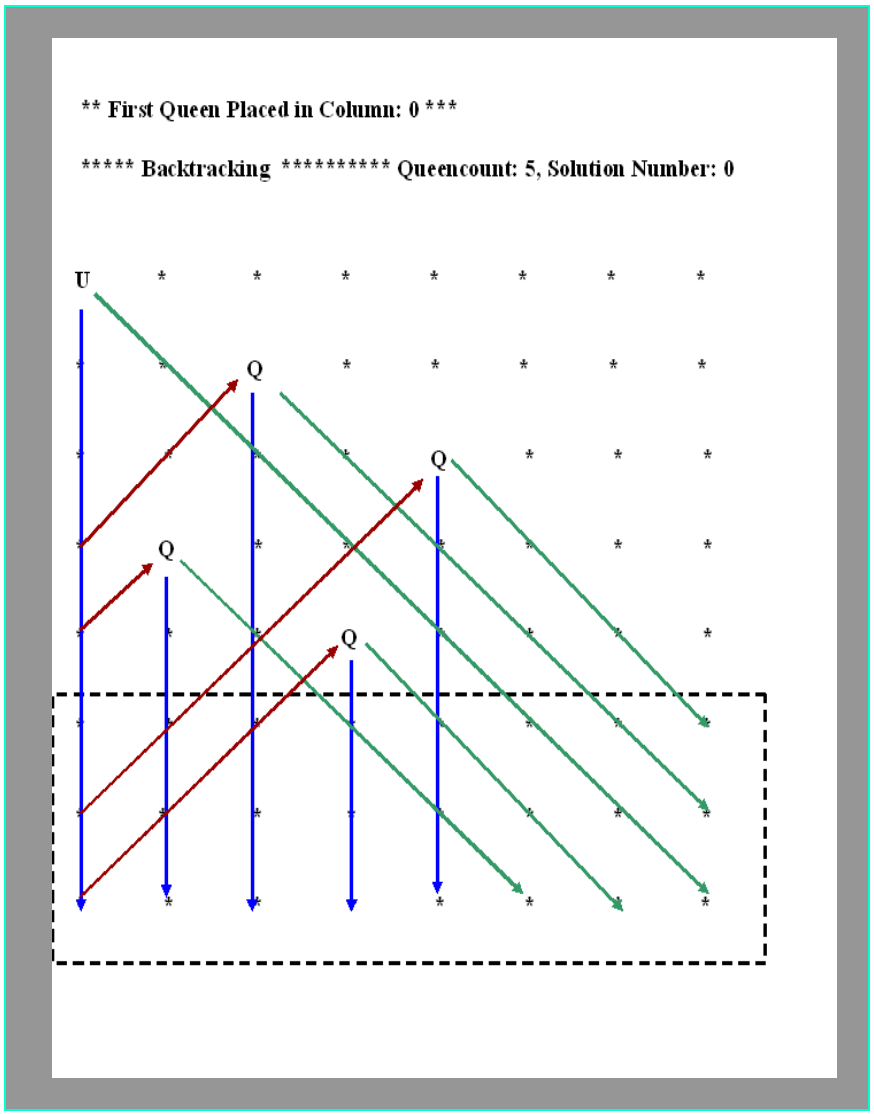
The queen number n is also the row number;
Increment queen count;
Colfree [n] = *FALSE*;
Downfree [1 – n + 7] = *FALSE*;
Upfree [1 + n] = *FALSE*;

- When a **queen** is removed from a column ***n*** of a row the following resetting takes place:

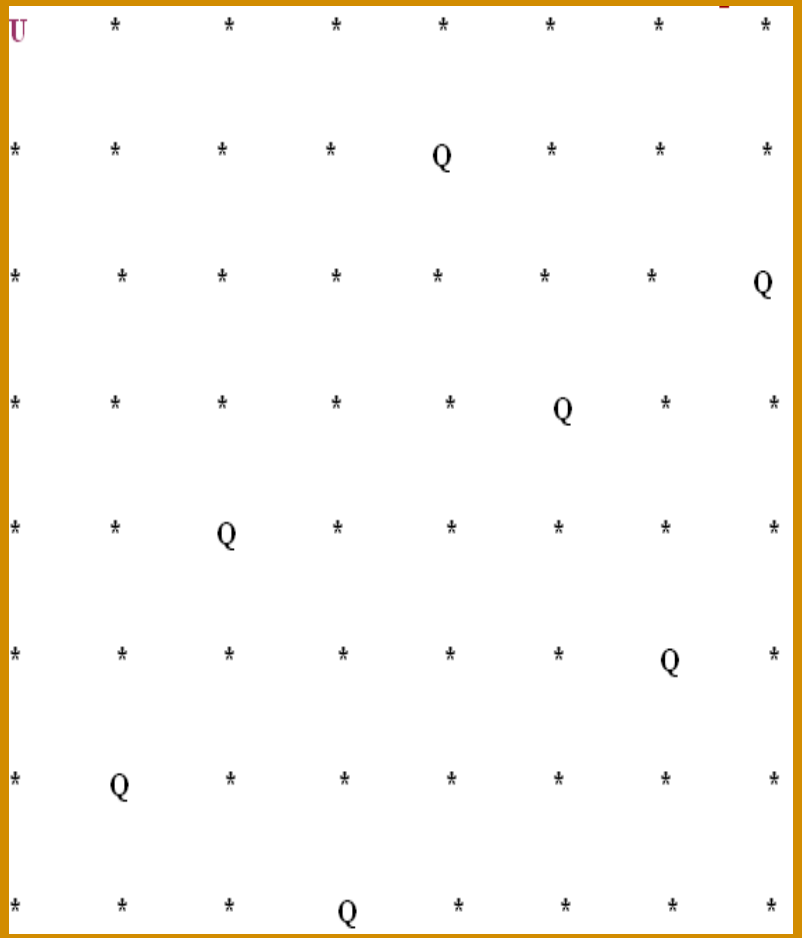
Decrement queen count;
Colfree [n] = *TRUE*;
Downfree [1 – n + 7] = *TRUE*;
Upfree [1 + n] = *TRUE*;

BACKTRACKING

BACKTRACKING SITUATION

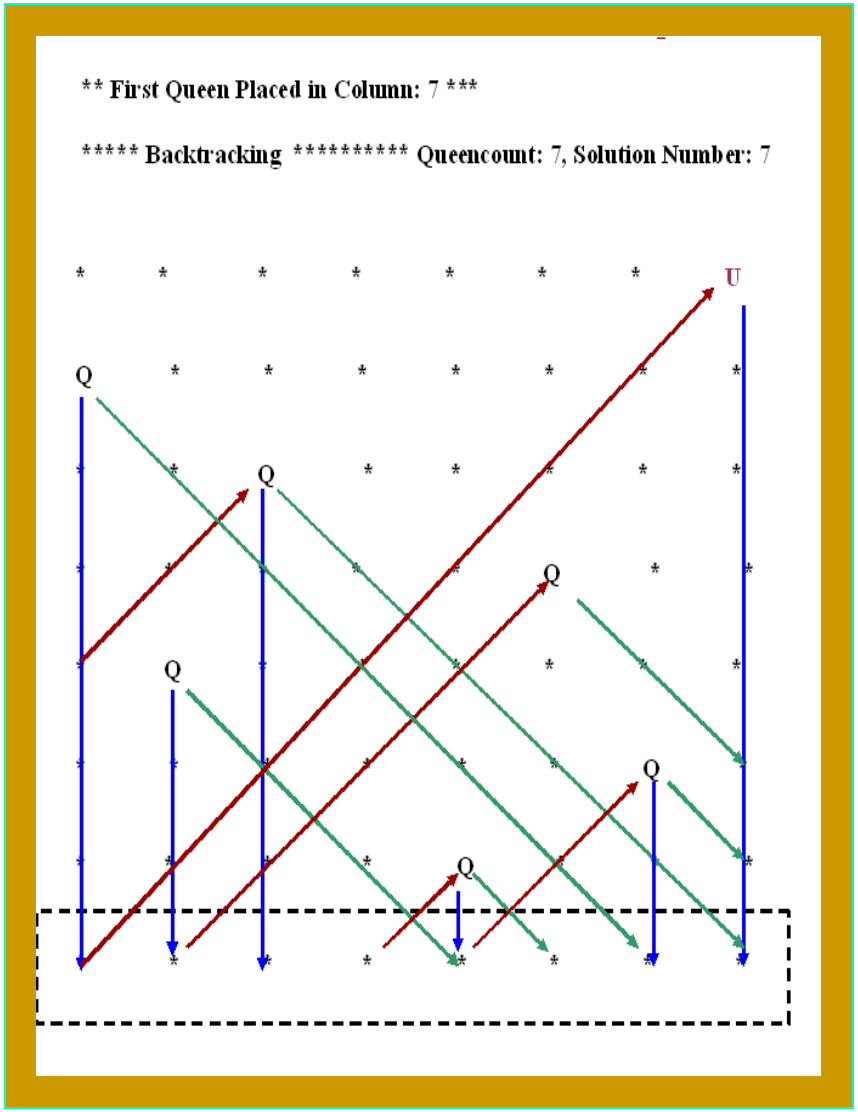


Solution for queen in column 0 after back tracking

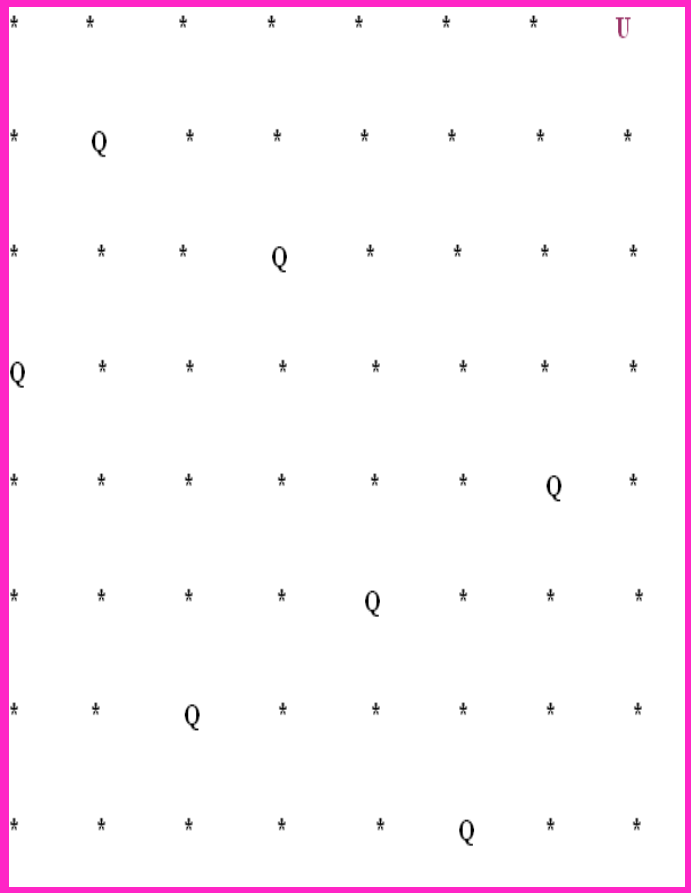


BACKTRACKING

BACKTRACKING SITUATION



Solution for queen in column 7 after backtracking



BACKTRACKING

EIGHT QUEEN SOLUTION PROGRAM

```
#include <stdio.h>
#define BOARDSIZE  8
#define DIAGONAL (2 * BOARDSIZE - 1)
#define DOWNOFFSET 7
#define FALSE 0
#define TRUE  1

void  writeBoard (void);
void  clearBoard (void);
void  addQueen   (void);

int   queencol  [BOARDSIZE]; /* queen column  */
bool  colfree   [BOARDSIZE]; // is column free
bool  upfree    [DIAGONAL ]; //  up diagonal free
bool  downfree [DIAGONAL ]; //down diagonal free
int   queencount = -1; // row queen is placed
int   numsol     =  0;  // number of solutions found
```

```
int  main (void)
{
        int  i;
        for (i = 0;  i < BOARDSIZE;  i++)
        {
            clearBoard ();
            queencol[++queencount]  = i;
            colfree[i]    = FALSE;
            upfree[queencount + i] = FALSE;
            downfree [queencount  - i +
                        DOWNOFFSET] = FALSE;
            addQueen   ();
        }       /* end of for loop */
        return 0;
}       /* end of main  function  */
```

BACKTRACKING

EIGHT QUEEN SOLUTION PROGRAM

```
void addQueen (void)
{
    int  col;  /*  column being tried for the queen  */
    queencount++;

    for (col = 0; col < BOARDSIZE; col++)
    {
       if (colfree [col] && upfree [queencount + col]
             &&  downfree [queencount - col +
                                   DOWNOFFSET ] )
      { // put the queen in position (queencount, col)
          queencol [queencount]       = col;
          colfree  [col]                  = FALSE;
          upfree   [queencount + col] = FALSE;
          downfree [queencount - col +
                          DOWNOFFSET] = FALSE;
```

```
          if (queencount == BOARDSIZE - 1)
          {    /* terminal condition */
               printf ("\n %d Queen Solution: %d
                              \n\n", 8, ++numsol);
               writeBoard ();
          }
          else
               addQueen ();   /* recursive call  */
          if (queencount - 1 < 0)
              continue;
          colfree [col] = TRUE; // backtrack queen
          upfree[queencount - 1 + col] = TRUE;
          downfree [queencount - 1 - col +
                          DOWNOFFSET] = TRUE;
          queencount--;
      }   /* end of if colfree        */
   }      /* end of for loop          */
}         /*  end of function addQueen */
```

BACKTRACKING

EIGHT QUEEN SOLUTION PROGRAM

```
void writeBoard (void) /* prints the output of N
        queens placement  */
{
        int  col;
        static  int  qcount = 0;
        for (col = 0;  col < BOARDSIZE;  col++)
        {
            if (queencol [qcount] == col)
                printf ("Q     ");
            else
                printf ("*     ");
        }       /* end of for loop  */
        printf ("\n\n");
        if (qcount++ < BOARDSIZE - 1)
           writeBoard ();
        qcount = 0;
}       /* end of writeBoard function  */
/* clears the board  for next placement */
```

```
void clearBoard (void)
{
        for (int i = 0;  i < BOARDSIZE;  i++)
        {
            colfree    [i] = TRUE;
            queencol [i] = -1;
        }       /* end of for loop  */
        for (int j = 0;  j < DIAGONAL;  j++)
        {
            upfree     [j] =  TRUE;
            downfree [j] =  TRUE;
        }       /* end of for loop  */
        queencount = -1;
}       /* end of clearBoard function  */
```

BACKTRACKING

EIGHT QUEEN SOLUTION PROGRAM

User placed first queen in col 0 :

U * * * * * * *
* * * * Q * * *
* * * * * * * Q
* * * * * Q * *
* * Q * * * * *
* * * * * * Q *
* Q * * * * * *
* * * Q * * * *

User placed first queen in col 1:

* U * * * * * *
* * * Q * * * *
* * * * * Q * *
* * * * * * * Q
* * Q * * * * *
Q * * * * * * *
* * * * * * Q *
* * * * Q * * *

User placed first queen in col 2 :

* * U * * * * *
Q * * * * * * *
* * * * * * Q *
* * * * Q * * *
* * * * * * * Q
* Q * * * * * *
* * * Q * * * *
* * * * * Q * *

User placed first queen in col 3:

* * * U * * * *
Q * * * * * * *
* * * * Q * * *
* * * * * * * Q
* Q * * * * * *
* * * * * * Q *
* * Q * * * * *
* * * * * Q * *

BACKTRACKING

EIGHT QUEEN SOLUTION PROGRAM Continued

User placed first queen in col 4:

```
*  *  *  *  U  *  *  *
Q  *  *  *  *  *  *  *
*  *  *  Q  *  *  *  *
*  *  *  *  *  Q  *  *
*  *  *  *  *  *  *  Q
*  Q  *  *  *  *  *  *
*  *  *  *  *  *  Q  *
*  *  Q  *  *  *  *  *
```

User placed first queen in col 5 :

```
*  *  *  *  *  U  *  *
Q  *  *  *  *  *  *  *
*  *  *  *  Q  *  *  *
*  Q  *  *  *  *  *  *
*  *  *  *  *  *  *  Q
*  *  Q  *  *  *  *  *
*  *  *  *  *  *  Q  *
*  *  *  Q  *  *  *  *
```

User placed first queen in col 6:

```
*  *  *  *  *  *  U  *
Q  *  *  *  *  *  *  *
*  *  Q  *  *  *  *  *
*  *  *  *  *  *  *  Q
*  *  *  *  *  Q  *  *
*  *  *  Q  *  *  *  *
*  Q  *  *  *  *  *  *
*  *  *  *  Q  *  *  *
```

User placed first queen in col 7:

```
*  *  *  *  *  *  *  U
*  Q  *  *  *  *  *  *
*  *  *  Q  *  *  *  *
Q  *  *  *  *  *  *  *
*  *  *  *  *  *  Q  *
*  *  *  *  Q  *  *  *
*  *  Q  *  *  *  *  *
*  *  *  *  *  Q  *  *
```