The Eight Queens Problem

Task:

Place 8 queens on a 8x8 chessboard so that no queen checks another. Two Queens check each other if,

- **They are in the same row,**
- **They are in the same column,**
- **They are on the same diagonal.**

Cost of Computation for N-Queens

Place N queens on an NxN board.

2x2 -- No Solution

3x3 -- No Solution

- **Choose N squares from N^2 square.**

  i.e. $\binom{N^2}{N}$

  for N=8, Ans = $4.4 * 10^9$ (approx)

- **One Queen per row.**

  N choices for Queen 1

  N choices for Queen 3

  ...

  Total = N^N

  for N = 8, Ans = $1.7 * 10^7$, approx

- **and as well, one Queen per column,**

  N Choices for Queen 1

  N-1 Choices for Queeen 2

  ...

  Total= N!

  for N = 8, Ans = $4 * 10^4$, approx

Representations used in program

As in Rooks problem we use an array to represent a solution.

q : ARRAY[INTEGER]

where q.item(i) denotes the column position of the i-th queen on the i-th row. This guarantees one queen per row.

<u>Checking Cols</u> (As in Rooks problem).

Used_Col : ARRAY[BOOLEAN]

By default all items are set to False.
Used_Col(i) iff there is a Queen on Col i.
This prevents two queens being on the same column.

<u>Diagonals</u>.

There are 2N-1 diagonals in each direction.
An Up_Diagonal starts bottom-left and and ends top-right.
All squares (i,j) in an up-diagonal have sum i+j.
A Down_Diagonal starts top-left and end bottom-right.
All squares (i,j) in a down-diagonal have the same difference i-j.
There are 2N-1 Up_Diagonals and 2N-1 Down_Diagonals.

Let

Up_Diag : ARRAY[BOOLEAN]
Down_Diag : ARRAY[BOOLEAN]

<u>where</u>
Up_Diag.item(k) iffa queen is on Up_Diag.item(k), and Down_Diag.item(k) iff a queen is on Down_Diag.item(k)

e.g.
The squares (3,5) (2,6) are on the Up_Diag.item(8)
and squares (3,7) (4,8) are on Down_Diag .item(-4)

Find All Solutions to the N-Queens Problem

```
All_Queens(i : INTEGER) is
        local
                j : INTEGER
        do
                if i > q.size then
                        Print_Queen
                else
                        from
                                j := 1
                        until
                                j > q.size
                        loop
                                if safe(i,j) then
                                        Set_Queen(i,j)
                                        All_Queens(i+1)
                                        Reset_Queen(i,j)
                                end
                                j := j+1
                        end -- loop
                end
        end -- All_Queens
```

A position (i,j) is safe if it is not checked by any previously placed Queen, i.e. it neither on the same Column nor the same Up_Diagonal nor the same Down_Diagonal as a previous Queen.

In Setting a Queen on position (i,j) we set the arrays Used_Col, Up_Diag and Down_Diag such that

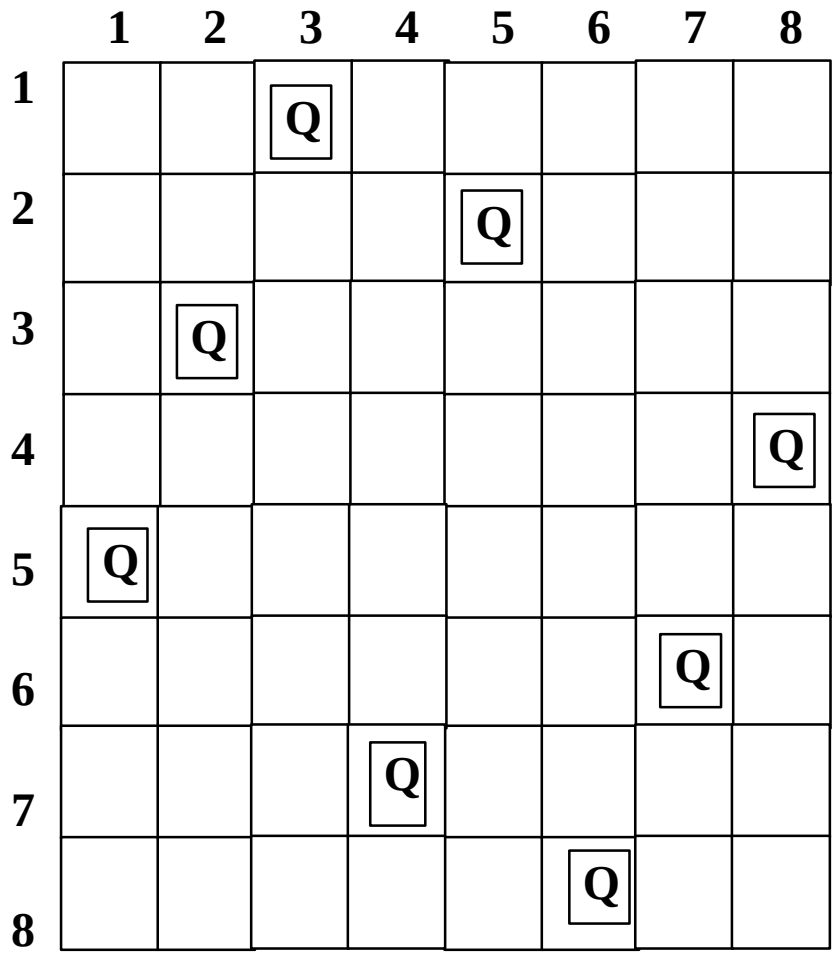
Used_Col .item(j) = True
Up_Diag.item(i+j) = True

and Down_Diag .item(i-j) = True,
i.e. in Resetting we undo the Setting.

The class for All Solutions to N-Queens

The class includes routines for printing a solution both in array and matrix form. The solutions could be redirected to a file.

There are 92 solutions for N = 8, but only 12 are essentially unique in that they are not symmetries of each other. There are 8 symmetries of the board and so there should be 8*12 solutions in total. There are only 92 as some solutions in the 96 are symmetries of each other.
Consider Solution: 3 5 2 8 1 7 4 6



This solution contributes only 4 to the total of 92.There are only 3 other symmetries of this solution.

4 6 8 2 7 1 3 5
5 3 1 7 2 8 6 4
6 4 7 1 8 2 5 3

<u>Exercise</u>: Write a procedure to find the 'first' solution of the N-Queens problem.

```
class GEN_QUEENS
creation
        make
feature
        q : ARRAY[INTEGER]

        Used_Col : ARRAY[BOOLEAN]
--Used_Col.item(k) iff there is already a queen on col k

        Up_Diag : ARRAY[BOOLEAN]
-- Up_Diag .item(k) iff a queen already on Up_Diag.item(k)

        Down_Diag : ARRAY[BOOLEAN]
-- Down_Diag.item(k)  iff  a queen on Down_Diag.item(k)

        counter : INTEGER
        -- counter for solutions

        Mat : ARRAY2[CHARACTER]
        -- for storing solution to N-Queens
```

```
make is
        local
                N : INTEGER
        do
                io.put_string("%N Enter size ")
                io.get_int
                N := io.last_int
                !!q.make(1, N)
                !!Up_Diag.make(2, 2*N)
                !!Down_Diag.make(-(N-1), N-1)
                !!used_Col.make(1, N)
                !!Mat.make(N,N)
                Mat.initialize(‘ ‘)
                io.put_string("%N The Solutions are: %N")
                counter := 0
                All_Queens(1)
        end -- make
```

```
safe(i,j : INTEGER) : BOOLEAN is
        do
                result := not(Used_Col.item(j)
                                or else Up_Diag.item(i+j)
                                or else Down_Diag.item(i-j))
        end -- safe

Set_Queen(i,j : INTEGER) is
        do
                q.put(j,i)
                Used_Col.put(True,j)
                Up_Diag.put(True,i+j)
                Down_Diag.put(True,i-j)
        end -- Set_Queen

Reset_Queen(i,j : INTEGER) is
        do
                Used_Col.put(False,j)
                Up_Diag.put(False,i+j)
                Down_Diag.put(False,i-j)
        end -- Reset_Queen
```

```
All_Queens(i : INTEGER) is
        local
                j : INTEGER
        do
                if i > q.count  then
                        Queens2Matrix
                        Print_Matrix
                        Clear_Matrix
                else
                        from
                                j := 1
                        until
                                j > q.count
                        loop
                                if safe(i,j) then
                                        Set_Queen(i,j)
                                        All_Queens(i+1)
                                        Reset_Queen(i,j)
                                end
                                j := j+1
                        end -- loop
                end
        end -- All_Queens
```

```
Queens2Matrix is
    local
        i,j : INTEGER
    do
        from
            i := 1
        until
            i > q.count
        loop
            from
                j := 1
            until
                j > q.count
            loop
                if q.item(i) = j then
                    Mat.put('Q',i,j)
                end
                j := j+1
            end
            i := i+1
        end
    end -- Queens2Matrix
```

```
Print_Matrix is
    local
        i,j : INTEGER
    do
        io.new_line
        Print_Queen
        io.put_string("%NIn Matrix form: %N")
        from
            i := 1
        until
            i > Mat.height    -#rows
        loop
            from
                j := 1
            until
                j > Mat.width  -- #columns
            loop
                io.putchar(Mat.item(i,j))
                io.putchar(' ')
                j := j+1
            end
            io.new_line
            i:=i+1
        end
        io.new_line
    end -- Print_Matrix
```

```
Clear_Matrix is
    local
        i,j : INTEGER
    do
        from
            i := 1
        until
            i > Mat.height
        loop
            from
                j := 1
            until
                j > Mat.width
            loop
                Mat.put(' ',i,j)
                j := j+1
            end
            i:=i+1
        end
    end -- Clear_Matrix
```

```
        Print_Queen is
                local
                        k : INTEGER
                do
                        counter := counter+1
                        io.put_string(" Solution #")
                        io.put_integer(counter)
                        io.put_string(":  ")
                        from
                                k := 1
                        until
                                k > q.size
                        loop
                                io.put_integer(q.item(k))
                                io.putchar(' ')
                                k := k+1
                        end
                        io.new_line
                end -- Print_Queen


end -- GEN_QUEENS
```