Assignement name : n\_queens

Expected files : \*.c \*.h

Allowed functions : atoi, fprintf, write

-------------------------------------------------------------------------

Write a program that will print all the solutions to the n queens problem

for a n given as argument.

We will not test with negative values.

The order of the solutions is not important.

You will display the solutions under the following format :

<p1> <p2> <p3> ... \n

where pn are the line index of the queen in each colum starting from 0.

For example this should work :

$> ./n\_queens 2 | cat -e

$> ./n\_queens 4 | cat -e

1 3 0 2$

2 0 3 1$

$> ./n\_queens 7 | cat -e

0 2 4 6 1 3 5$

0 3 6 2 5 1 4$

etc...

NOTE: what is the N-queens problem?

The N-Queens problem is a classic algorithmic puzzle in computer science and discrete mathematics.

It asks: how can you place N chess queens on an N×N chessboard so that no two queens threaten each other?

According to some sources, this means no two queens can share the same row, column, or diagonal.

Here's a breakdown:

The Goal:

The objective is to find all possible configurations (or a single configuration) of queen placements that satisfy the constraint of no attacks.

The Constraints:

Queens attack horizontally, vertically, and diagonally.

Therefore, a valid solution requires that no two queens occupy the same row, column, or diagonal.

Complexity:

The N-Queens problem is known to be NP-complete for the decision problem (whether a solution exists), and also for the problem of finding a single solution.

Common Solutions:

Backtracking algorithms are frequently used to solve the N-Queens problem efficiently.

The problem can be represented as a set of permutations of the numbers 1 to N, where each number represents the row of a queen in a given column.

For example, for N=4, a solution might be represented as the permutation, which means:

Column 1 has a queen in row 2.

Column 2 has a queen in row 4.

Column 3 has a queen in row 1.

Column 4 has a queen in row 3.

The N-Queens problem is a good example of how seemingly simple rules can lead to complex algorithmic challenges.

It's also a problem with applications in areas like scheduling and resource allocation.