Overview

The goal of this assignment is to find solutions for the “8-queen” puzzle/problem. The goal is to place on a 8x8 chess board 8 queens, in a way that none of the queens is able to Threaten any other queen using the standard chess moves for a queen (row-wise, column-wise, and diagonally).

8-queen Solution(s) and Recursion

There are 92 possible solutions for this puzzle. However not all of them are unique. Some of the solutions are identical after rotating or mirroring the board. If we count these solutions as one, there are 12 unique solutions for the problem.

Suppose initially we have the board all cleared up. Each queen is taking one column namely A, B,…, H. We adopt the “algebraic chess notation” to denote the board position shown on figure 1.

![Figure 1](image)

We place exactly one queen at each column. A convenient way to reference the queens is Q1, Q2,…, Q8 , which simply denote the queen positioned at the 1st, 2nd, …, 8th column.

The algorithm for finding a solution can be described as following:
1. We can place Q1 queen on any row in the first column. Let’s choose position A1.

2. The second queen goes on the second column. Starting from the bottom up we notice there will be a threat on B1 and B2. Therefore, the first available position is B3. Now we have to restrict the remaining 6 queens from being placed on rows 1 and 3, and diagonals A1-H8 and A2-G8. The board after placing the second queen is shown in figure 2. The red positions are the restricted ones.

3. The positioning of the remaining queens can take place following similar considerations. However after placing Q7 at G6 we will end up with the following situation graphically shown in figure 3. There will be a threat to the Q8 if it is placed at any of the positions on the column 8 (figure 3).

4. Luckily we have the opportunity to revise the position of already placed queens in order to find an acceptable position for Q8. We start
backtracking from the last placed queen, Q7. The remaining positions G7 and G8 will not be safe for Q7, therefore we must proceed by relocating Q6 from its current position occupying F4. Once we find a new safe position for a queen, we can continue to the next column, and look for a different solution starting from the next column on just as we did when we started to solve the problem. If you proceed in this manner, the first occurrence of a solution will be one corresponding to the board shown in figure 4.

As you see, in this case, it was necessary to backtrack all the way to the second column and relocate Q2 from its initially assigned position (you may try this).

5. Once a solution is reached, the result can be printed on the screen. Then we can search for additional solutions. As an idea how to do this is the following. Since Q8 is the last positioned queen we go ahead and consider other available positions for it, which are H4-H8 in figure 4. In case of threats we backtrack in the usual way as described before.

In developing a program to solve the 8-queens problem, the board can be encoded in many ways. A convenient way is to represent the positions of the queens on the board with a one dimensional array. For example the solution in figure 4 can be represented as:

```
1 7 4 6 8 2 5 3
```
The Programming Assignment and Requirements

The assignment is to write a recursive program that implements backtracking to solve the 8 queen puzzle. The accepted solutions are board arrangements, where there are no threats between the queens.

Requirement 1: User input and Output
Your code has to allow user input. The input allows user to specify the positions for the first n queens, where 0<n<8, in the format described in the previous section. If the user specifies the positions of interest for the first n>0 queens, your program should start looking for solutions starting from the column n+1. If there is already a threat between the queens with the input supplied by the user, the program reports that a solution is impossible that includes the specified input. Otherwise, the program look for possible solutions starting by placing the queens in the n+1 and subsequent columns and report all solutions (if any) which match the user’s specification.

For example, to indicate that we want solutions, in which the first three columns are taken by the queens placed on the second, fourth, and sixth rows, the user will do the following:

>Input the initial state: 2 4 6 0 0 0 0 0

In this case, your program shall find all solutions (if any) for the remaining five columns. Hence the positions specified with zero “0” in the user input denote the empty columns.

For output, your program should print all possible solutions according to the format shown below:

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

...............................................................
Requirement 2: Functions

Your program must implement the two functions described below.

1. Function `nextColumn`

This function should accept an array input, corresponding to a board that has valid positions for the first n queens on the first n columns, and it returns an integer for the first valid position (if any) for the queen on the next column, n+1 (starting from row 1 and ending with row 8). By valid we mean that no threats exist between the queens. If none can be found this function returns 0.

As an example, consider the board in figure 5:

![Figure 5](image.png)

This board can be specified as \( b = [1 \ 3 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \). Since the first zero in the array \( b \) appears at index 2 of the array, i.e. the 3rd column of the board (array indices start from 0 in C). If the function `nextColumn(b)` is called, it will return 5 since this is the first position starting from row 1 where a queen can be placed without any threats from the other queens.

2. Function `nextPosition`

Now assume we are not interested in 5th row position for to be occupied by our 3rd queen. If the function `nextPosition(b)` is called with the value of \( b = [1 \ 3 \ 5 \ 0 \ 0 \ 0 \ 0 \ 0] \) it should find (if any) and report the next possible (valid) row for the last queen in its column (in this example the 3rd
col.). If no more valid position can be found for the last queen in its column, this function returns 0 (recall that the order is from rows 1 through 8 ascending). Therefore, in the current example, the call `nextPosition(b)` returns the integer value 6. This means that the 6\textsuperscript{th} row is the next valid position for the 3rd queen. This function accepts an array input, corresponding to a board with valid positions for the first n queens on the first n columns, and it returns an integer.

**Extra Credit (25 points)**

Not all valid solutions to the 8-queen problem are unique. Some are reflections or rotations of other solutions. For example all the solutions shown in figure 6 are not unique. The second solution is a reflection and the third solution (from left) is a rotation of the first solution.

In order to receive the extra credit given here, implement the function `isUnique` and use it in your program to only report unique solutions for the given 8-queen problem. The function `isUnique` accepts two arrays `b1` and `b2` corresponding to two board configurations and returns 1 if they are unique and 0 if they are not. Therefore:

\[
\text{isUnique}([2,4,6,8,3,1,7,5],[5,7,1,3,8,6,4,1]) = 0
\]

Your program implementing the extra credit reports unique solutions of the given problem similar to the following format.

\[
\begin{array}{cccccccc}
\text{US} & 2 & 4 & 6 & 8 & 3 & 1 & 7 & 5 \\
\end{array}
\]