Author’s Name: Mubasshir Al Shahriar (Nihal)

# Assignment: The n queens problem using dynamic memory.

Short Report/Summary:

* This program is an updated version of the previous 1D version (with print) of 8 queens problem’s assignment where we can see the total number of solutions not only 8-queens problem, rather we can use custom number of queens while running the code. Here in stead of typical static memory, we are using dynamic memory. We are solving the n-queens puzzle inside a function named “nqueens” which is an integer function that returns (not prints) the total number of available solutions for n-queens problem. We’re using almost the same algorithm which we’re using in every 8-queens assignment. here we still using the code in 1D format with 2 while loops.
* Here we still using the code with same algorithm in 1D format with 2 while loops where the basic algorithm is: We have “n” number of queens to put in the chessboard so that no multiple queens are in same row column or diagonal. Starting from the first row, first column, the program goes to the next column of the chessboard through outer while loop. Then, if the program has already passed the last column, that means it has found one solution (which was not found before in this program). SO, it uses “solutionsnum++” to add 1 to the total number of solutions for n-queens. Then starts backtracking to start searching for another solution. Otherwise, it enters the inner while loop, starting from the first row of that column, it calls the previously created ok function, that runs 3 tests (row test, up diagonal test, and down diagonal test) and checks if it is ok to place a queen in that row. If it says the position is ok (returns true), then the program goes back to the beginning of the outer loop and prints the result. Otherwise, inside the inner loop, it just goes to the next row and keeps checking the position with ok function until it finds a suitable position, or it reaches the last row of the column (n-1). In case it reaches the last row of the column, yet couldn’t find a place, it backtracks and goes to the previous column to change the previous position of queen in that column. The outer while loop gets terminated when it successfully finds all the possible solutions and then the function comes to an end returns “solutionsnum” which is the total number for solutions for n-queens problem.
* Inside the main function, we used a for loop and used the “nqueens” function (which contains the above algorithm) inside it. How many times the for loop going to be iterated/ “nqueens” functions will be called depends on the value of “n” set by the user. Based on that “n,” the “nqueens” function gets called and returns the total number of solutions for each of them. Like: if the value of “n” is set to 3, then “nqueens” function will be called 3 times and first it will return the number of solutions available for 1-queen problem, then 2-queens problem, then 3-queens problem. And this function only returns the number of solutions but doesn’t print. Also, in this problem we didn’t use any print function or any code to print the any structure of the found solution. This code finds complete solution but doesn’t print them in the output.
* The main point of this project is: we needed to dynamically allocate memory as the memory needed here is dependent on user input (n). Inside the “nqueens” function, we declared the array of size n dynamically ( int\* q = new int[n]; ). This is the main and most important part of this assignment. We used the "new" operator to request the “heap” to dynamically allocate the memory. Then just normally followed the regular algorithm and completed just like written in the 2nd point/paragraph. Each time this “nqueens” function runs and finds all the possible solutions, just before returning the total number of solutions (solutionsnum), we used the delete [] operator to deallocate q each time. It deletes the array and allows to free up the memory to make it available for future use. This is a great way to prevent memory leak.

Comment: This is my short report to solve the “: The n queens problem using dynamic memory” project. So, now we are not only bound to 8-queens, we can use any number while running the code. We dynamically allocated the array and after our job is done, we deallocated as well. Thich shows us a great way to prevent memory leak. I compiled and ran the code; it ran successfully and showed correct numbers of solutions in the output.

Screenshots of Output:  
   
  


