

**Lab report**

|  |  |
| --- | --- |
| **Course**: | Class Libraries and Data Structures |
| **Semester**: | 1st semester of the academic year **2018-2019** |
| **Major**: | Software Engineering |
| **Class**: |  |
| **Student Name**: |  |
| **Student ID:** |  |
| **Teacher:** | ZHAO, Hengjun (赵恒军) |

**School of Computer and Information Science**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | | Recursion and Backtracking | | | |
| Date | | Nov 2, 2018 | Type | | √ Confirmatory  √ Design  □Comprehensive |
| 1. **Objective & Requirements**    1. Familiar yourself with the basic idea of recursion    2. Be able to design your own recursive functions. The key is to design the basic case and the induction step    3. Understand the backtracking algorithm. Can solve a specific problem using recursion and backtrack. | | | | | |
| 1. **Experimental environment (**platform and software**)**   Windows 7 (or higher versions) + Visual Studio 2010 (or higher versions) | | | | | |
| 1. **Experimental content and design** (Main Content, Procedure, Codes and Results)   **Task 1**  Develop and validate a program to place eight queens on a chess board in such a way that no queen is under attack from any other queen.  Procedure:   1. Define a class called eightqueen. I put all methods in it including the recursive one. 2. Create a default chessboard by constructor. Then define methods printchess() and goBack() which are easier ones. 3. The most important methods are isValid() and findPosition().   But firstly, I need to think carefully about how to put 8 queens in ***main() function***. It turns out that the easiest method is to put queens row by row starting from chess[0][0], since there must be 1queen in every row,.   1. As a result, in isValid(), I only check “the rows and columns above” the position given by parameters r and c (i.e. chess[r][c]), namely: every column of the row r, top left diagonal and top right one. Since every time I would put 1 queen in every row, there is no need to check rows for chess[r][c]. Moreover, the default return values should be 1, as all occasions except mentioned three above are valid. 2. In method findPosition(), the fist step should be a check for position chess[r][c], after which, if it is valid it could turn to find a possible position in the next row.   For searching a possible queen in the next row and considering that every column might have one, I use a loop to traverse every column checking if there is a valid position, that is, backtracking findPosition() with parameters *r+1,i* **(***i is* the column in the loop**)**. In other words, I call this method for every row to search for a queen. If there is a valid position, put 1 in chess[r+1][i]. On the contrary, if there is no valid queen in the last row **(i.e.** until the position chess[7][7] is not valid**)**, it should goBack() which means the last position--chess[r][c] is wrong. **(**The r+1 is transferred to findPosition() so when it is returned, the last position is chess[r][c]**)**  It is also need to mention that since I transfer the parameter r+1 to find a queen in the next row, the end of the findPosition() should be **r+1<size**, which is before the check of every column.   1. The last thing we need to do is coding main(). Create an object for class eightqueen and call findPosition(0,0) with corresponding output. Then everything is done.   Code:          Result:    Below is the process of outputting 92 solutions:   1. To find the position and records them and considering that there must be a queen in each row, I set row as an array with its values represents the valid column. Also, the chessboard (a[9][9]) and numbers of solutions(t) are the extern variables. 2. For literally convenience, I set chessboard as an array with 9 columns and rows. In Judge(), it is also much easier to use abs()—the absolute function to find the diagonal positions.In findQueens(), there is a recursive call in order to find a position in the next row. If it returns void, which means there is no valid position in next row, there would be 0 for every position.   CODE:            RESULT:  (Due to its long length I put the code and .exe considering convenience) | | | | | |
| 1. **Result analysis and discussion**（Analysis of experimental results and summing up the harvest and the existing problems）   The sequence of check isValid() and backtracing and how to find a position in every line simultaneously is difficult. I have been wondering for a long time for the sequence of these two since the end of the backtrcking and goBack() decides the position of findPosition(). It turns out the simplest way to understand backtracking isfinding the end of the recursion first, and then return the needed value. It means, coding the stop of the recusion before finding the valid position.  For finding the possible position, my method is to traverse every column. Although it is not an efficient one, it is still a secured way to find a possible queen. And my method only could find one solution starting from chess[0][0]. | | | | | |
| Comments & Evaluation | Content & Design (A-E) | | |  | |
| Procedure & Codes (A-E) | | |  | |
| Results (A-E) | | |  | |
| Analysis & Discussion (A-E) | | |  | |
| Score (A-E):  Feedback comments: | | | | |