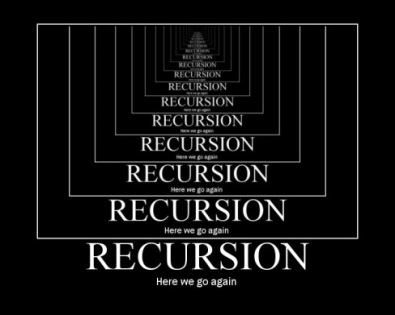
**CS273 ASSIGNMENT #4: Problem Solving with Recursion**

## MY NAME: Aaron Borjas DUE: 11/15

**Grade:**

|  |  |  |
| --- | --- | --- |
| **CATEGORY** | **POINTS** |  |
| **EX4\_1: Binary Search** |  | 40 |
| **EX4\_2: Eight-Queens Problem** |  | 40 |
| **TOTAL** |  | 80 |
| **EX4\_3: Advanced Eight-Queens Problem (extra credit)** |  | 40 |

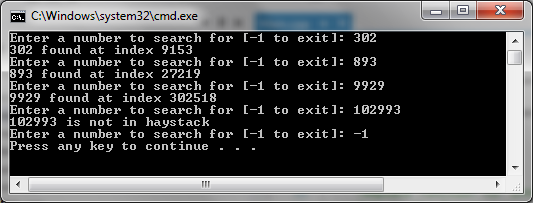
**Recursion is tough! But we can only get better with practice. For this homework assignment, you are asked to implement the recursive solutions to two problems we studied in class: searching for an item in a sorted vector, and the Eight-Queens problem.**

**To begin, please download (clone) the three VS Code projects from the RecursionHomework repository to your desktop.**

## EX4\_1

In the **RecursionTutorials** folder, you will find a folder called **EX4\_1**. Modify the Visual Studio project in this folder to implement the template recursive **binary search** function. Read **section 7.3**, the power-point slides for **the recursion lecture**, and **the recursion study guide**. (Matt will post this study guide on GoogleDrive.)

To verify that your binary search implementation is correct, your program should look like this when it is run:



**Save your solution in your directory on CS1 and submit via WhitGit.**

## EX4\_2

In the **RecursionTutorials** folder, you will find a folder called **EX4\_2**. Modify the Visual Studio project in this folder to implement the recursive solution of the **Eight-Queens** problem in the function **PlaceQueen**. Read the power-point slides for **the recursion lecture** and **the recursion study guide**.

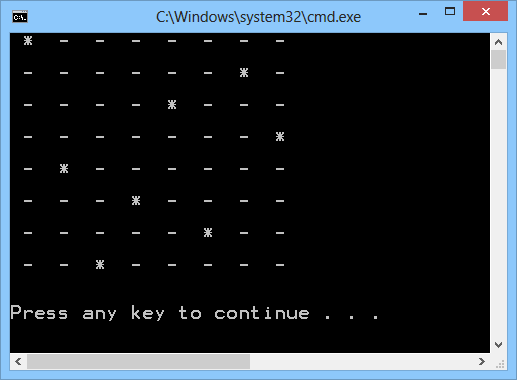
The recursive implementation assigns a queen, one column at a time, up to the 8 queens that needs to be placed on the 8x8 board. Note that because no two queens can be placed on the same row, every row in a successful placement will have one queen assigned to it (after all there are 8 rows and 8 queens).

Much of the implementation has been done for you. The primary data structure used in this implementation is the **RowPlacement** vector. This vector will be used to hold the assignment of a queen in each column to a row. For example, assigning a queen in column 7 to row 4 is **RowPlacement[3] = 6** (assuming row and column 1 starts with index 0).

Our implementation already provides you with a **ValidPosition** function that checks if a previous column queen assignment has a diagonal that prevents you assigning the current column queen to a specific row.

Finish the implementation of the **PlaceQueen** function.

This is how your program should run if you are successful:



**Ditto for submission.**

## EX4\_3EC (extra credit)

In the **RecursionTutorials** folder, you will find a folder called **EX4\_3EC**. Modify the Visual Studio project in this folder to implement the recursive solution to the **Eight-Queens** problem as before, but this time, allow your code to generate all **92 possible solutions** to the problem (i.e. there are 92 possible arrangements of queens in your 8x8 chess board that satisfy the problem).

Note that we have declared a vector to store every successful 8 queen placement:

vector< vector<int> > PlacementSolutions;

As you find successful solutions, you can add that **RowPlacement** vector into this as follows:

PlacementSolutions.push\_back(RowPlacement);

**Ditto on submission.**