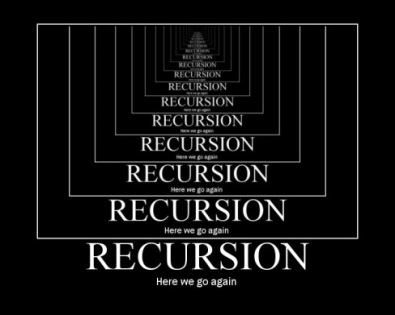
**CS273 ASSIGNMENT #8: Problem Solving with Recursion**

## MY NAME: Enerelt Bat-Erdene

**Grade:**

|  |  |  |
| --- | --- | --- |
| **CATEGORY** | **POINTS** |  |
| **EX8\_1: Binary Search** |  | 50 |
| **EX8\_2: Eight-Queens Problem** |  | 50 |
| **TOTAL** |  | 100 |
| **EX8\_3: Advanced Eight-Queens Problem (extra credit)** |  | 10 |
| **EX8\_4: Maze solver (extra credit)** |  | 20 |

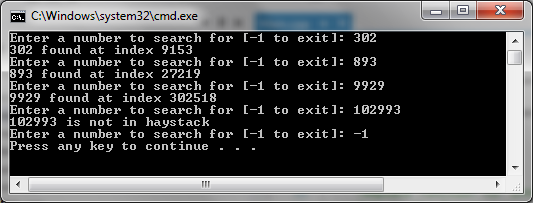
**Recursion is tough! But we can only get better with practice. For this homework assignment, you are asked to implement the recursive solutions to two problems we studied in class: searching for an item in a sorted vector, and the Eight-Queens problem.**

**To begin, please copy** [**Recursion**](file:///\\CS1\CS_ClassData\273\RecursionTutorials\)**Tutorials.zip from our class folder to your desktop.**

## EX8\_1

In the **RecursionTutorials** folder, you will find a folder called **EX8\_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 **lecture 21**, and **study guide 21**.

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



**Save your solution to your GitHub account**

## EX8\_2

In the **RecursionTutorials** folder, you will find a folder called **EX8\_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 **lecture 21** and **study guide 21**.

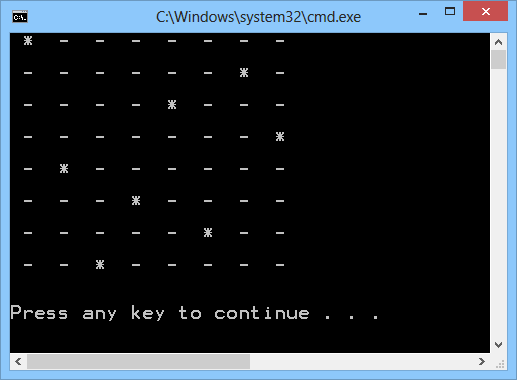
The recursive implementation assigns a queen, one column at a time, up to the 8 queens that needs to be placed on the 8x8 board. Note that because no two queens can be placed on the same row, every row in a successful placement will have one queen assigned to it (after all there are 8 rows and 8 queens).

I have done most of the implementation 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).

My 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:



**Save your solution to your GitHub account**

## EX8\_3 (extra credit)

In the **RecursionTutorials** folder, you will find a folder called **EX8\_3**. 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);

**Save your solution to your GitHub account**

## EX8\_4 (extra credit)

In the slides for this session, you will find discussion on solving a maze. A maze can be represented as a two-dimensional array, where each cell in the array indicates which routes are available for the next move. Implement a solution that reads an arbitrary maze and outputs its solution. Note, the RecursionTutorial.zip file does not contain any code to get you started (which explains why it's worth more extra credit points).

**Save your solution to your GitHub account**