This problem is derived from Programming Problem 11 Page 391 of the text.

Suppose that you have n queens from a chessboard. Is it possible to place all n queens on the board so that no two queens are in the same row, no two queens are in the same column, and no two queens are on the same diagonal? For example, a solution with n = 5 is shown here:

```
Stack Pre-Test Passed!


Please enter n for the n-queens problem (n >= 1): 5
Solution for n = 5
=========
- - - Q -  Queen in row 1, column 4
- Q - - -  Queen in row 2, column 2
- - - - Q  Queen in row 3, column 5
- - Q - -  Queen in row 4, column 3
Q - - - -  Queen in row 5, column 1
=========
Program ended with exit code: 0
```

This problem is called the n-queens problem. For this project, you are to write a function that has one integer parameter, n, and determines whether there is a solution to the n-queens problem. If a solution is found, then the procedure prints the row and column of each queen. Your program should solve the problem by making a sequence of choices, such as “Try placing the row 1 queen in column 1,” or “Try placing the row 7 queen in column 3.” Each time a choice is made, the choice is pushed onto a stack that already contains all the previously made choices.

The purpose of the stack is to make it easy to fix incorrect choices, using the following pseudocode, with a stack, s, and a boolean variable, success:

while (!success && !s.empty( )) {

Check whether the most recent choice (on top of the stack) is in the same row, same col- umn, or same diagonal as any other choices (below the top). If so, then we say there is a conflict; otherwise there is no conflict.
if (there is a conflict)

//Pop elements off the stack until the stack becomes empty, or the top of the stack is a choice that is not in column n. If the stack is now not empty, then increase the column number of the top choice by 1.

else if (no conflict, and the stack size is n)

//Set success to true because we have found a solution to the n-queens problem.

else

//Push information onto the stack, indicating that the next choice is to place a queen at row number s.size( )+1, and column number 1.

}

This technique is called backtracking since we keep our choices on a stack and back up to correct any mistakes that are made. Notice that when you check for a conflict, you will need access to the entire stack (not just the top), so that you should use the peek function.

This project is three files: stack3.h

Download stack3.h

, stack3.tpp.h

Download stack3.tpp.h

, and queens.cxx

Download queens.cxx

The stack must be implemented as a Hardware stack. That is a push() will grow down through the array. And a pop() will move up through the array.

Do not change the function prototypes in stack3.h. You will need to adjust some on the implementation code. Also, implement the needed functions in stack3.tpp.h. And implement the needed code in queens.cxx.

Extra Credit Option (Easy):

Convert your stack implementation to be Linked List based. Your main does not change. These changes are in stack3.h and stack3.tpp.h. You cannot use helper function nor node.xxx from previous assignments. The code is very simple and you can put the definition of a node into stack3.

Submission Requirement:

Run with 3, 5 and 8 queens. Append results into one file

Upload your:

1. .cxx file(s).
2. stack3.h
3. stack3.tpp.h
4. output.rtf (w/3 runs in it)