Computer Engineering 171
Homework 2: Object-Oriented Programming

Due: October 26th at 9:00 am

In this assignment, you will use C++ and Smalltalk, two object-oriented languages discussed in class. Use the Linux machines in the Engineering Computing Center for this assignment. The Smalltalk interpreter is `gst` (you will need to execute "`setup smalltalk`" first) and the C++ compiler is `g++`.

1 Binary Search Trees

Under `/scratch/coen171/homework2`, you will find an implementation in Smalltalk of a binary search tree. Translate this program to C++ using templates, keeping the same class and operation names. A small test program is also provided, which will become the function `main` in C++.

Goal: To learn about classes as a mechanism for information hiding.

2 The 8-Queens Problem

In chess, a queen can move any number of squares horizontally, vertically, or diagonally. The ***8-queens problem*** is the problem of trying to place eight queens on an empty chessboard in such a way that no queen can attack any other queen. This problem is intriguing because there is no efficient algorithm known for solving the general problem. Rather, the straightforward algorithm of trying all possible placements is most often used in practice, with the only optimization being that each queen must be placed in a separate row and column:

1. Starting with the first row, try to place a queen in the current column.
2. If you can safely place a queen in that column, move on to the next column.
3. If you are unable to safely place a queen in that column, go back to the previous column, and move that queen down to the next row where it can safely be placed. Move on to the next column.

In the course folder, you will find my solution to the 8-Queens Problem in the file `Queens.cpp`. My solution provides the following generalizations:

1. Solves the n-Queens Problem rather than the 8-Queens Problem. The chessboard is of size $n \times n$ rather than of size 8×8, and n queens are to be placed on the board.
2. Computes and displays the ***total*** number of solutions possible. Rather than finding one solution and exiting, my program will find all possible solutions and write the total number of possible solutions to standard output.
3. Solves the n-***Pieces*** Problem rather than the n-Queens Problem. Given any kind of chess piece, not just a queen, my program displays the total number of solutions possible for placing n of those pieces on the board such that no piece can attack any other piece.

You will need to write a file called `Queens.h` that contains the class and function definitions for each chess piece. Write a base chess piece class with which my algorithm works. The base class should provide at least the following methods:

- `int row()`: returns the row number of this piece
- `int column()`: returns the column number of this piece
- `void place(int row, int col)`: places this piece on the board given a row number and a column number
- `bool menaces(const Piece *p)`: given another piece, returns true if this piece can attack the given piece, and false otherwise

Now write classes for each of the following chess pieces:

- rooks: A rook can move any number of squares horizontally or vertically, but cannot move diagonally.
- bishop: A bishop can move any number of squares diagonally, but cannot move horizontally or vertically.
- knight: A knight can move two spaces in either a horizontal or vertical direction and an additional space in the other direction.
- queens: A queen can move any number of squares horizontally, vertically, or diagonally. Thus, it can move as either a rook or a bishop.
- amazon: An amazon can move either as a queen or as a knight.

Use virtual multiple inheritance in your solution. Prove that your solution is correct my filling in the missing values in the table at the top of `Queens.cpp`.

Goal: To generalize a problem using object-oriented techniques.

Hints: Write and test each piece in the order listed.