ET-580 Separate Compilation Homework

Reading

1) **Chapter 10 Pointers and Dynamic Arrays**

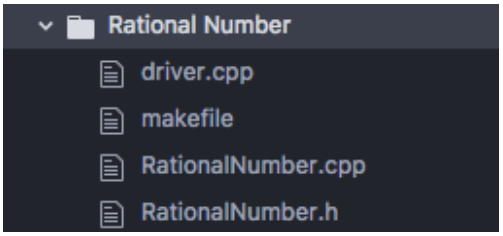
Implementation

1) Rational Number

Implement your Rational Number homework from Operator Overloading using separate compilation.

1. Duplicate your original homework into a new folder.
2. Separate definitions from declarations into .h and .cpp files.
3. Separate your driver program into a separate file called driver.cpp.
4. Use conditional compilation preprocessor directives in your .h file.
5. Implement a makefile to run your project.

Your directory should look like this:



Submission

1. Use "make clean" to remove all .o files and your application file.
2. Zip the contents and name them with your name.
3. Upload to blackboard.

2) Solving the N-Queens Problem

Main Function

Create a one-dimensional array of size n and seed it with -1 values. This represents an empty n x n chess board. Develop the problem with n=4, test the solution up to n=24.

Note that the array index represents the row, and array value the column.

Print Function

Print the array where all -1 values are empty spots on the board and all values not equal to -1 represent the position of a queen.

isSafePosition

Test if a specific row/col position is safe for a queen relative to all previous rows (row-1). If placing a queen into this specific row/col position would be dangerous return false, else return true.

Solve

Use backtracking to provide a solution for the current board size. This is a recursive function which repeatedly tests different queen layouts until it ultimately finds a working solution.

Your implementation should be based off of the following design:

```
 7 › void output(const int a[], const int &SIZE) {…}
19
20 › bool isSafePosition(const int a[], const int &SIZE, int row, int col) {…}
41
42 › bool solve(int a[], const int &SIZE, int row) {…}
75
76 ⌄ int main() {
77       cout << endl;
78
79       // create an array and populate it with -1 to represent an empty board
80       const int SIZE = 8;
81       int a[SIZE];
82       for(int i=0; i<SIZE; i++) { a[i] = -1; }
83
84       // solve the recursive problem
85       solve(a, SIZE, 0);
86
87       cout << endl;
88       return 0;
89   }
```