1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Homework07, put each programming exercise into its own subdirectory of this directory, zip the entire Homework07 directory up into the file Homework07.zip, and then submit this zip file to Homework #7.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyCLasses.

All non-templated class structures are to have their own guarded specification file (.h) and implementation file (.cpp) that has the same name as the class. All templated class structures are to be guarded and written entirely in their (.h) file. No inline coding in the class specification. In addition you must create a make file that compiles and links the project on a Linux computer with a Debian or Debian branch flavor.

2 Programming Exercises

2.1 Program #1

You may recall that Pascal's triangle from your previous mathematics classes. The entries in Pascal's triangle are called combinations since if we want to know how many ways to select k objects from a set of n objects we simply go to the n^{th} row and k^{th} column of the triangle and read off the number. The row and column numbers start at 0. So to find out how many ways you can select 2 objects from a set of 5 we go to row number 5 and column number 2 and see that there are 10 ways to choose 2 from 5. We denote n choose k mathematically as

$$\begin{pmatrix} n \\ k \end{pmatrix}$$

1					
1	1				
1	2	1			
1	3	3	1		
1	4	6	4	1	
1	5	10	10	5	1

1	6	15	20	15	6	1				
1	7	21	35	35	21	7	1			
1	8	28	56	70	56	28	8	1		
1	9	36	84	126	126	84	36	9	1	
1	10	45	120	210	252	210	120	45	10	1

You may also recall that the way to get any entry in the middle of the triangle you simply take the sum of the entry above the one you are calculating and the entry one above and one to the left of the one you are calculating. So in mathematical terms,

$$\begin{pmatrix} n \\ k \end{pmatrix} = \begin{pmatrix} n-1 \\ k \end{pmatrix} + \begin{pmatrix} n-1 \\ k-1 \end{pmatrix}$$

Of course, if n or k is 0 then the value is 1, if $n = k$ the value is 1, and we will define anything outside these ranges the value is 0. That is, if n or k are negative or if $k > n$ we will simply have the program return 0. Write a recursive program to calculate n choose k. A couple runs are below. There are formulas using factorials for this calculation but you are not to use those, use the recursive definition given above.

```
Enter n and k (integer) with a space between them: 5 2
5 choose 2 = 10

Enter n and k (integer) with a space between them: 10 4
10 choose 4 = 210

Enter n and k (integer) with a space between them: 6 3
6 choose 3 = 20

Enter n and k (integer) with a space between them: 12 12
12 choose 12 = 1

Enter n and k (integer) with a space between them: 15 16
15 choose 16 = 0

Enter n and k (integer) with a space between them: 15 7
15 choose 7 = 6435
```

2.2 Program #2

Update the code for the 8 Queens problem that we went over in class to find all the solutions to the problem. Have the program print out just the row positions for each solution, hence you will not need the matrix or chessboard files, just the STL vector. The output just for starting in position 0 is below. Also have the program print out the number of solutions to the problem.

```
0 4 7 5 2 6 1 3
0 5 7 2 6 3 1 4
0 6 3 5 7 1 4 2
0 6 4 7 1 3 5 2
...
```

2.3 Program #3

Update the code for the general 8 Queens problem you did in the last exercise and generalize it to an $n \times n$ chess board. The user will input n. An example of a program run is below.

```
Input the board size n X n. n = 5
0 2 4 1 3
0 3 1 4 2
1 3 0 2 4
1 4 2 0 3
2 0 3 1 4
2 4 1 3 0
3 0 2 4 1
3 1 4 2 0
4 1 3 0 2
4 2 0 3 1

The number of solutions is 10
```

The beginning of a run with $n = 9$ is below.

```
Input the board size n X n. n = 9
0 2 5 7 1 3 8 6 4
0 2 6 1 7 4 8 3 5
0 2 7 5 8 1 4 6 3
0 3 1 7 5 8 2 4 6
0 3 5 2 8 1 7 4 6
0 3 5 7 1 4 2 8 6
...
```

2.4 Program #4

In class we discussed the Towers of Hanoi puzzle and its recursive solution. We will extend this puzzle to one using 4 pegs instead of three. As you may suspect the 4 peg version can be solved in fewer moves since you will have two temporary pegs instead of just one.

Before we get into the 4 peg version take the code for the three peg version and add in the ability for the hanoi function to count the number of moves that have been made. This

is probably easiest done by adding an integer parameter to the function, passed by reference, and each time a move is made increment the counter.

Now create another recursive function called hanoi4 that will do the 4 peg version. Here is how the 4 peg version works. If the number of disks is 0 then there is nothing to do, just return from the function. If the number of disks is 1, just move from the starting peg to the ending peg. If the number of disks is 2 or more then first solve the problem for $n-2$ disks (yes, $n-2$ not $n-1$) from the start peg to the first temporary peg, using the second temporary peg and the end peg as the first and second temporary pegs respectively. Then do three moves, start peg to second temporary peg, start peg to end peg, then the second temporary peg to the end peg. Then solve the problem again on $n-2$ disks from the first temporary peg to the end peg using the start peg and the second temporary peg as the first and second temporary pegs respectively. Also include the move counting like you did with the three peg version.

Write a main that will ask the user for the number of disks, have the program print out the list of moves for both the 3 peg and 4 peg versions along with the number of moves for each. Runs on 3, 4, and 5 disks are below. In each of these we have the starting peg as A, the ending peg as B and the temporary peg as C for the three peg version and pegs C and D for the four peg version.

```
Enter the number of disks: 3
The solution for n = 3 disks. Using 3 pegs.
Move A to B
Move A to C
Move B to C
Move A to B
Move C to A
Move C to B
Move A to B
Number of moves in the solution = 7

The solution for n = 3 disks. Using 4 pegs.
Move A to C
Move A to D
Move A to B
Move D to B
Move C to B
Number of moves in the solution = 5
```

```
Enter the number of disks: 4
The solution for n = 4 disks. Using 3 pegs.
Move A to C
Move A to B
Move C to B
Move A to C
Move B to A
Move B to C
Move A to C
Move A to B
Move C to B
Move C to A
Move B to A
```

```
Move C to B
Move A to C
Move A to B
Move C to B
Number of moves in the solution = 15

The solution for n = 4 disks. Using 4 pegs.
Move A to B
Move A to C
Move B to C
Move A to D
Move A to B
Move D to B
Move C to D
Move C to B
Move D to B
Number of moves in the solution = 9
```

```
Enter the number of disks: 5
The solution for n = 5 disks. Using 3 pegs.
Move A to B
Move A to C
Move B to C
Move A to B
Move C to A
Move C to B
Move A to B
Move A to C
Move B to C
Move B to A
Move C to A
Move B to C
Move A to B
Move A to C
Move B to C
Move A to B
Move C to A
Move C to B
Move A to B
Move C to A
Move B to C
Move B to A
Move C to A
Move C to B
Move A to B
Move A to C
Move B to C
Move A to B
Move C to A
Move C to B
Move A to B
Number of moves in the solution = 31

The solution for n = 5 disks. Using 4 pegs.
Move A to D
Move A to B
Move A to C
Move B to C
Move D to C
Move A to D
Move A to B
```

```
Move D to B
Move C to A
Move C to D
Move C to B
Move D to B
Move A to B
Number of moves in the solution = 13
```