Computer Science II — Homework 3 — Pointers and Classes

This assignment is due Thursday, February 8 at 11:59:59pm and is worth 80 points toward your homework grade. In solving this homework you may use any C++ construct or algorithm you wish from Lectures 1-6, but not beyond. The homework is in two parts.

Part 1: Practice with Pointers and Dynamic Memory

Write a short program that reads in a file of integers, sorts them, and outputs to a second file the min, max, median and mode, all on separate lines. The mode is the integer than occurs most often. If two or more integers tie for the mode — e.g. 15, 27 and 44 all occur three times and no other integer occurs more than twice — then all of these values should be output and all should be on the same line.

There are two restrictions to this part of the assignment. First you must use arrays and dynamic memory allocation — no vectors. This means you need to figure out how many values are in the input automatically. (One way to do this is to read the file twice. This isn't the most efficient, but it will do for this assignment.) Everything must be done using pointer arithmetic and pointer dereferencing. There should be no integer loop counter variables.

In short the program should run with the command line

```
hw3_part1 input-integers stats-out
```

You may adapt as much or as little of the example code from Lectures 1-6 as you wish. You do not need to write any functions. Example input and output will be posted on the course web page.

Part 2: Matrix

The Sudoku game has rapidly become popular all over the world in the past few years. Part of this popularity has been fueled by the availabilty of sophisticated computer programs that automatically test generate games, evaluate their difficulty, check their correctness, and provide hints.

In case you do not know how Sudoku works, here is a brief explanation. Given is a positive integer, usually $n = 3$, and this integer is used to form an $n^2 \times n^2$ grid, usually 9×9. At the start, some of these $n^2 \times n^2$ grid locations have numbers in them — the numbers may be any of the values in the set $\{1, 2, \ldots, n^2\}$. For example,

	7		5			8		
	4		3	7			9	6
3		2			8	1		
		6		4			1	5
	9		2		7		3	
7	3			8		9		
		5	4			3		7
2	8			5	9		4	
		7			1		5	

The requirement is that each row, each column, and each of the $n \times n$ non-overlapping blocks (starting with the upper left corner), must have no repeated values. Thus, in the upper right hand corner, there can not be a 1, 6, 8 or 9 because these values are already in the same 3×3 block. Moreover, there can not be a 5 or 7 in this corner, because these values are already in the same row (and coincidentally in the same column). At this point it appears to be possible to put a 2, 3, or 4 here — we can not tell just from the constraints we have discussed thus far. (These three tests will be referred to as the *row*, *column* and *grid-block* tests.) We can determine that in this same block, just below the 9, a 7 must be placed, resulting in

	7		5			8		
	4		3	7			9	6
3		2			8	1	**7**	
		6		4			1	5
	9		2		7		3	
7	3			8		9		
		5	4			3		7
2	8			5	9		4	
		7			1		5	

In order to see why there must be a 7 placed here, consider the other possibilities:

- 1, 2, 3, 8 are eliminated by other entries in the same row (row 2),
- 1, 3, 4, 5, 9 are eliminated by the other entries in the same column (column 7).
- 1, 6, 8, 9 are eliminated by the other entries in the same 3×3 grid block, and

7 is the only possibility left.

Your job in Part 2 of HW 3 is to design, implement and test a class to represent a Sudoku grid and to provide a checking on the validity of placing certain numbers in that grid based on the row, column and grid-block tests. By no means do you have to solve the entire Sudoku problem. We are providing you with the start of a class declaration in a file, `Sudoku.h`. You **may not change** the declarations of the public member functions in this class, although you may add functions. You must of course provide private member variable(s), and you must implement the member functions to meet the design requirements specified in the file. You should, of course, write a main function that tests your class implementation, but you do not need to submit. We will test your implementation against our own main function, the start of which — not the complete version — will be posted on the course website. **It is crucial that you do not change the prototypes of the class member functions we have provided.**

In order to be consistent, here are a few conventions:

- Row numbering will run from 0 to $n^2 - 1$ (e.g. from 0 to 8 when $n = 3$).
- Column numbering will run from 0 to $n^2 - 1$, as well.
- The *nxn* blocks will be indexed as well, with block rows running from 0 to $n - 1$, and block columns running from 0 to $n - 1$.

   - For example, when $n = 3$, block $(1, 2)$ is the middle block on the right hand side, with upper left corner being grid entry $(3, 6)$ and lower right corner being grid entry $(5, 8)$.

What to Submit

For part 1, submit a single program file called `hw3_part1.cpp`.

For part 2, submit two files:

- `Sudoku.h`, which is your extension of our header file declaring the `Sudoku` class, and
- `Sudoku.cpp`, which is your implementation the `Sudoku` class member functions.

We have provided initial implementations of the `hw3_part2.cpp` and, of course, `Sudoku.h` files on the course web site. The latter contains a complete description of the functionality of the Sudoku class.

When you submit your solutions to this assignment, you will get feedback on the results of part 1 and on the results of part 2 using just the main program we are providing on-line. Our grading tests, especially for part 2, will be more extensive.

You do not need to submit a +readme.txt+ file unless there are special grading issues you would like to raise.