3: Squares and the Board

CSCI 4526 / 6626 Spring 2014

1 Goals

- To use default constructors and constructors with parameters.
- To use an input file to construct an array of objects.
- To model the Sudoku board.

2 The Board Class

A traditional sudoku board is a 9 by 9 array of squares. Sometimes we want to view this array as two-dimensional but at other times, we want to view it as a flat array. Various data structures could be used to represent this array, but we will use a simple flat array of 81 Squares and provide the 2-D subscripting through a subscript function.

Normally, an array of 9 things would be accessed using subscripts 0. . . 8. For Sudoku, this turns out to be endlessly confusing, so the player will use 1-based subscripting instead. This means that every player input (1–9) must be converted to a 0-based value (0–8) before using it. Further, the player will input both row and column numbers. Our program will take these two 1-based subscripts and combine them into a single 0-based subscript. This is a nuisance, but better than the alternative (a confused player who thinks the interface is ugly).

To do. Add another pair of files to your project to implement a Board class. Put the following parts into Board. (More parts will be added next week.)

- `#include` commands for `"tools.hpp"` and `"Square.hpp"` must be at the top of Board.hpp.
- `#include "Board.hpp"` must be at the top of Board.cpp. No other include commands belong in the cpp file.
- A private array of 81 Squares named bd. (NOT pointers to Squares.)
- A private `ifstream` that will be used to read in the data for a puzzle.
- A destructor that prints a trace comment.
- A constructor: `Board(const char* filename);` Print a trace comment. Open the named file. (Call `fatal()` if this step fails.) The input file will have 9 lines, each consisting of 9 data characters and a newline. A data character is a dash or a digit 1..9. Execute a nested loop for j =1..9 and k = 1..9 to:
  - Read the next character from the file.
  - If it is a data character, construct a Square using that character and the row and column subscripts, j and k. Store that square in the next slot of the array `bd`. Use j and k and the subscript function, below, to calculate that slot number.
  - After storing the square in bd, if k equals 9, read another character and verify that it is a newline.
  - After creating 81 squares, verify that you are at the end of the file. Do this by attempting to read one more character and testing for eof.
  - If any step finds an error in the input file format, call fatal. (Soon, we will replace the calls on fatal by throwing exceptions.)

- A subscript function, `Square& sub( int j, int k )` that uses the 2D square coordinates to compute and return a reference to Square object in the Board's array. The formula is: $(j-1)*9+(k-1)$
- A print function that prints all of the squares on the board, one per line, with a blank line after every 9th square. Delegate the task of printing a square to the print function in the Square class.
- Outside the class but inside the .hpp file, declare an inline method for the output operator:
  `inline ostream& operator<< (ostream& out, Board& b);`
  It must call your *print()* function with the appropriate stream parameter and return the `ostream&`.

Hints:

- You do NOT need dynamic allocation for anything.
- Use this syntax inside the loop to create a Square object and copy it into the proper position in Board's array: bd.sub(j, k) = Square(input, j, k);
- After debugging on-screen, send your output to a file so you can turn it in. At this time, there is too much output to use cut-and-paste techniques.

3 Due October 3

Construct a test plan for Board. (Assuming that the Square class works properly.) Incorporate this test plan into a function called `testBoard()`. Change `main()` to call both `testSquare()` and `testBoard()`.

Make a folder for turning in your work. The name of the folder should include both your name and the problem number. This problem is Sudoku-3. (Example: S3-Fischer). Put copies of your test plans, your source code (.cpp and.hpp files) and your output into this directory, then zip it up. Use email to my home to turn in your zip file. It is very important to turn this in on time.