3: Squares and the Board

CSCI 4526 / 6626 Fall 2019

1 Goals

- To model the Sudoku board.
- To learn how to write and use a subscript function.
- To use default constructors and constructors with parameters.
- To use an open input file, passed as a parameter.

A traditional sudoku board is an 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 $0 \ldots 8$. For Sudoku, this turns out to be endlessly confusing, so the player will use 1-based subscripting instead. This means that every player input for a row or column number (1–9) must be converted to a 0-based value (0–8) before using it. The player will input both row and column numbers. Your 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).

2 The Board Class

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 "tools.hpp"` and `"Square.hpp"`.
- `#include "board.hpp"` must be at the top of Board.cpp. No other include commands belong in the cpp file.
- Data members:
  - An integer, N, the size of the puzzle. It come into the module as a parameter to the constructor.
  - A private array of N^2 Squares named bd. (NOT pointers to Squares.) (Do NOT give this object a long name.)
  - A private `ifstream&` that will be used to read in the data for a puzzle. This open stream will be passed from Game to the Board constructor. The local stream variable must be initialized to the parameter value in a ctor.
  - A private short int `left`, used to track the number of squares left with a '-' instead of digit.
- Function members:
  - A destructor that prints a trace comment.
  - A constructor: `Board(int n, ifstream& puzfile);`
    Use ctors to initialize data members. Print a trace comment.

- `getPuzzle()`, a private helper function called from the constructor. The ifstream will contain N lines of input. Each line consists of N data characters and a newline. A data character is a dash or a digit between 1 and N. This function will read and process the N lines, and create N^2 squares. See detailed description in the next section.
- 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&`.

`getPuzzle()`
Execute a nested loop, $1 \ldots N$ to initialize the squares: On each iteration, read the next line of data from the stream and process it as follows:

- Loop through the chars on this line, k = $1 \ldots N$
- If the current char is a digit or a dash, construct a Square using that char and the row and column subscripts, j and k. Do not use new. Assign that square to the next slot of the array `bd`. Use j and k and the subscript function, below, to calculate the subscript for bd.
- After storing the square in bd, if k==9, verify that the next character is a newline.
- After creating N^2 squares, verify that you are at the end of the file. Do this by attempting to read one more character or line 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.)
- Hint: 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);

3 Changes to the Game Class

Now that the board class exists, it is possible to finish the game constructor. From the type code, determine whether the size of this puzzle is 6 or 9. Dynamically allocate a Board of the appropriate size, 6 or 9. (Test this only with the traditional puzzles, since the DiagBoard and SixyBoard classes will not exist for another couple of weeks.)

4 Due September 24

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

Hint: 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.

Submission: 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.