CSCI 241 Assignment 3
100 points

Purpose

This assignment covers two-dimensional arrays.

Overview

Sudoku is a logic-based, combinatorial number-placement puzzle. The rules of Sudoku are quite simple. In a traditional sudoku puzzle (and there are variations) the objective is to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 sub-grids that compose the grid (also called "boxes", "blocks", "regions", or "sub-squares") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which typically has a unique solution.

An example of a Sudoku puzzle

In order for a solution to a Sudoku puzzle to be correct, the following conditions must hold true:

1. Every row should contain the digits 1 to 9 but should not repeat the digits 1 to 9 at any point within that row.

2. Every column should contain the digits 1 to 9 but should not repeat the digits 1 to 9 at any point within that column.

3. Every 3x3 sub-grid should contain the numbers 1 to 9 but should not repeat the numbers 1 to 9 at any point within that sub-grid.

Here are some examples of puzzle solutions:

A correctly filled Sudoku grid

An incorrectly filled Sudoku grid

Writing a computer program to solve Sudoku puzzles can be quite challenging, and typically uses algorithms and data structures outside the scope of an intermediate class in programming. However, writing a program that will **check** a solution to a Sudoku puzzle and verify that it is correct is a lot easier.

Program

Write a class called `Verifier` that can be used to verify whether a Sudoku puzzle solution is correct (i.e., whether or not it meets the conditions outlined above).

As with Assignment 2, the class definition for the `Verifier` class should be placed in a header file (`Verifier.h`) while the method definitions should be placed in their own source code file (`Verifier.cpp`).

Data members

The `Verifier` class should have a private data member to represent a Sudoku grid as a two-dimensional array of 9 rows, each with 9 columns. The exact data type of the array elements is up to you; you might decide to store the digits of the Sudoku grid as integers, or you may choose to store them as characters.

You may create other private data members for the class if you want to.

Methods

The `Verifier` class should have the following public methods. You are welcome to create additional private methods for the class if you want to.

- `Verifier` default constructor - This constructor has no parameters. It should set all of the elements of the grid array to 0.

- `readGrid()` - This method takes one parameter: a pointer to a constant character (data type `const char*`), which will point to an array of characters that contains the name of a file to use as input. It returns nothing. The method should read the contents of the input file into the elements of the grid array.

  An input file contains exactly 81 numbers, arranged in 9 rows of 9 columns each, separated by whitespace. For example:

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

- `printGrid()` - This method takes no arguments and returns nothing. It should print the Sudoku grid array to the screen as 9 rows of 9 columns (the same way the grid appears in the input file). For example:

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

- `verifySolution()` - This method takes no arguments. It should return a Boolean value - true if the

Sudoku grid array contains a valid solution, false if not.

Driver Program

A driver program, `assign3.cpp`, is provided for this assignment. The purpose of a driver program is to test other pieces that you code. You do not need to write the driver program yourself. A copy of the driver program can also be found on `turing` at
`/home/turing/t90kjm1/CS241/Code/Fall2016/Assign3/assign3.cpp`.

```
/*****************************************************************
   PROGRAM:    CSCI 241 Assignment 3
   PROGRAMMER: your name
   LOGON ID:   your z-ID
   DUE DATE:   due date

   FUNCTION:   This program tests the functionality of the Verifier
               class.
*****************************************************************/

#include <iostream>
#include <string>
#include "Verifier.h"

using std::cout;
using std::endl;
using std::string;

#define NUM_FILES 7

int main()
   {
   Verifier v;
   string fileName;

   cout << "Sudoku Verifier\n";

   for (int i = 1; i <= NUM_FILES; i++)
      {
      cout << endl;

      // Construct file pathname
      fileName = string("/home/turing/t90kjm1/CS241/Data/Fall2016/Assign3/solution")
        + (char)('0' + i) + ".txt";

      // Read the solution file as input
      v.readGrid(fileName.c_str());

      // Print the Sudoku grid
      v.printGrid();

      // Verify whether or not the solution is correct
      if (v.verifySolution())
         cout << "\nThis is a valid Sudoku solution\n";
      else
         cout << "\nThis is not a valid Sudoku solution\n";
      }

   return 0;
```

```
    }
```

Implementation Hints

The driver program should not be modified for your final submission. But while you're developing, modifying the driver program can definitely be in your best interest. For example, you may want to take out the loop and concentrate on working with a single input file.

The input files used by the driver program are located in the directory

```
/home/turing/t90kjm1/CS241/Data/Fall2016/Assign3/
```

The files are named `solution1.txt` through `solution7.txt`. You can run the driver program on Unix without copying these files to your home directory, but you are welcome to make copies of them, modify the copies for testing purposes, download them to your local computer, etc. You can also easily create your own data files for testing.

The exact technique you use to verify whether a solution is correct or not is up to you, but it will most likely require you to loop through the columns of each row looking for duplicate values, loop through the rows of each column looking for duplicate values, and loop through the rows and columns of each sub-grid looking for duplicate values.

Other Points

- As always, programs that do not compile on `turing/hopper` automatically receive 0 points.
- Submit your program in the usual fashion.

Extra Credit (10 points)

Code your `printGrid()` method to print the Sudoku grid array in a slightly fancier form, similar to the following:

```
-------------------------
| 2 3 4 | 9 5 6 | 8 1 7 |
| 9 5 7 | 8 1 4 | 2 6 3 |
| 1 8 6 | 3 7 2 | 4 5 9 |
-------------------------
| 5 4 9 | 6 8 1 | 7 3 2 |
| 6 1 8 | 7 2 3 | 5 9 4 |
| 7 2 3 | 4 9 5 | 6 8 1 |
-------------------------
| 3 9 2 | 5 6 7 | 1 4 8 |
| 4 7 5 | 1 3 8 | 9 2 6 |
| 8 6 1 | 2 4 9 | 3 7 5 |
-------------------------
```