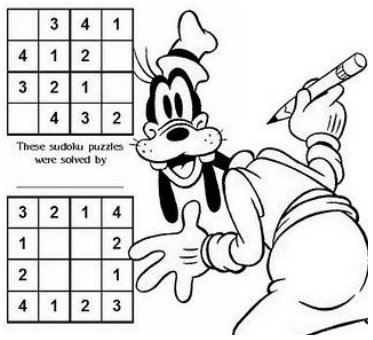
Project 5
Sudoku game:

SudokuBoard.h
SudokuBoard.cpp
sudoku.h
sudoku.cpp
test1.cpp



Due: **Friday, 9-April-2014 Due: 6:00 pm. Accepted until: 11:59:59 pm**

Grading: **60 points**
10 points: test1.cpp -- testing portion
40 points: SudokuBoard.cpp, sudoku.cpp -- runnability portion
10 points: "code quality" for all cpp files

Note: We grade your LAST submit only
Partners are allowed on this one.
If you work with a partner, BOTH must submit all portions
"Partners" is defined as 2 current students. Need NOT be in the same lecture/discussion section
Both partners need to put both names and uniqnames in all cpp files
Note: if you begin this project with a partner, you must finish it with the same partner.

Style grading: If you do this project with a partner, we will randomly choose one of the partner's to grade for style. Both partners will be given the same style score.

Note: Beware the autograder-cheat-detector magic machine
Further details are with this spec.

Learning Objectives

- To extend abilities in problem solving with a more complex problem in logic and design.
- To gain experience with classes, 2-dimensional arrays and file I/O.
- To have some fun along the way with a puzzle.

Overview of files

• `SudokuBoard.h` base project file that has prototypes of mandatory functions related to the SudokuBoard class	• `SudokuBoard.cpp` implementation file – 1 function is implemented for you
• `sudoku.h` base project file that has prototypes of mandatory functions	• `sudoku.cpp` implementation file – 1 function is implemented for you
• `ioutil.h` header file that has data definitions for "coloration"	• `ioutil.cpp` implementation of functions defined in "ioutil.h" (Do not modify this file.)
• `test1.cpp` holds "int main()" for testing for all functions	
• `board1.txt` sample board input file.	

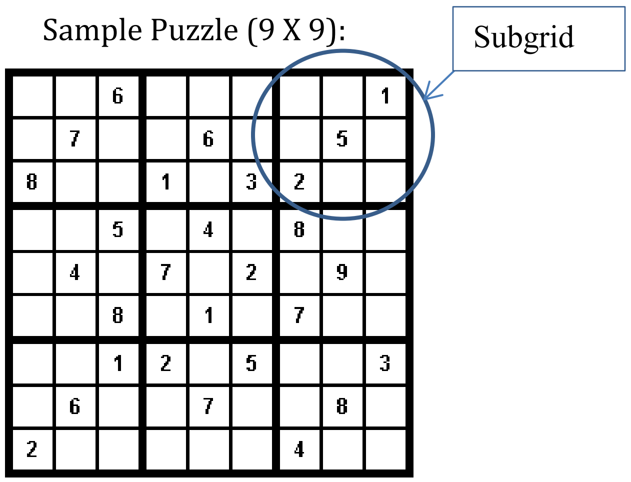
Overview

Sudoku is a combinatorial number-placement puzzle game. For a more detailed explanation of this game, see any of the following sites:

http://www.sudoku.com/
http://www.websudoku.com/
http://en.wikipedia.org/wiki/Sudoku

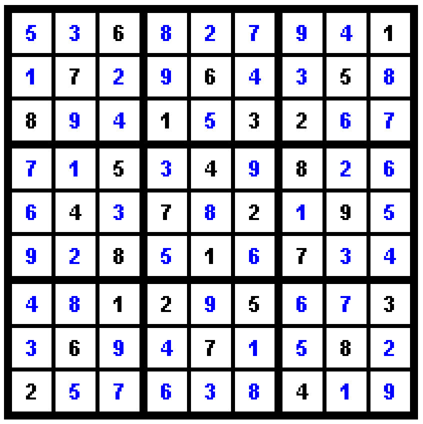
In most games of sudoku, the puzzle is a 9 X 9 grid. However, there are sudoku games that are played on 16 X 16 grids (hard) and 25 X 25 grids (obnoxious) or even 4 X 4 grids (really easy). The rules of the game are simple: make sure each row, column and subgrid (n x n) contains the numbers 1 through n exactly once. For a 9 X 9 game of sudoku (which is the most popular), the numbers 1 through 9 can appear once for each row, column, and subgrid. A move would consist of placing a value inside a blank spot. A sudoku puzzle would have some numbers filled in and some values blank (the less filled in, the harder it is):

Sample Puzzle (9 X 9):



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

Sample Solution:



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

Note that in the solution, the numbers 1 through 9 appear only ONCE for EACH row, column, and subgrid. Thus, any move that would lead to a number appearing more than once in a certain row, column, or subgrid is illegal.

Your task is to write a program that can read in a sudoku puzzle (in text form), allow a user to play it, and determine if they win. Your program does NOT have to generate puzzles, nor does the program have to solve them. The extent of strategy you need to know is what constitutes a legal move, an illegal move, and a win.

Approach to What should you do first

As you probably noticed, this puzzle deals with matrices, so arrays by themselves are probably not sufficient (although do-able). Instead, you need a 2-dimension array of structs:

```
struct Square
{
   int number;     // number is 0 if square is not occupied
   bool permanent; // true if permanent rather than user input
};
Square board[BOARD_SIZE][BOARD_SIZE] = {{0, false}};
```

The above code creates the 2-dimensional array of structs, which is essentially a matrix. This 2D array is declared in the `SudokuBoard.h` header file as a data member of a class called SudokuBoard. This class implements all gameplay functionality that is associated with a Sudoku board. You will need some auxiliary functions, which are declared for you in `sudoku.h`.

Testing/Implementation portion. This project consists of writing several functions according to specifications. All the RME's are within the GIVEN **`SudokuBoard.h`** and **`sudoku.h`** file. You are to implement these functions as given. Do not alter any of the prototypes; if you do, your code will not compile in the autograder. By convention, header files that declare a C++ class take on the exact same name as the class -- including any uppercase letters. Similarly, the .cpp file that implements the class should have the same name as the class. Therefore, the implementation code for `SudokuBoard.h` must be in a file named **`SudokuBoard.cpp`**. Similarly, the implementations of the `sudoku.h` functions must be in a file named **`sudoku.cpp`**. Be careful to pay attention to uppercase letters when you `#include "SudokuBoard.h"` and `#include "sudoku.h"` – any misspelling of uppercase characters will produce compile errors in the autograder.

The best approach is to implement one function and test it. You need to do this anyway so why not benefit from your efforts. Make sure your 1st function works perfectly before moving onto the next function. Implement the 2nd function and test it. Your code for testing functions should be in **`test1.cpp`** – this is also where `int main()` goes.

sudoku.h and sudoku.cpp

This is where you will want to begin. Declared constants and function declarations are in `sudoku.h`. All definitions should go in `sudoku.cpp`. There are three functions to implement and they are all related to I/O. You may use additional functions if you like, but those listed in sudoku.cpp are mandatory. As with the last two projects, these functions will be tested for adherence to spec. Note that **`void printCommands()`** is written for you.

SudokuBoard.h and SudokuBoard.cpp

These files will implement the bulk of the gameplay functionality. There are seven functions to implement and we have supplied **`void SudokuBoard::printBoard()`** for you. In many games the printing of the board is one of the most difficult functions. It can take considerable time to get it right -- and if you don't, the autograder will not like it. Therefore -- just use the one in `SudokuBoard.cpp`. All a other functions listed in `SudokuBoard.h`

This board prints in color and the "coloration" is done through `#include "ioutil.h"` with the corresponding `ioutil.cpp`. You will need to add `ioutil.cpp` into your project. Do not modify `ioutil.h` or `ioutil.cpp`.

Note: If on a Mac, or if ioutil.cpp is causing "issues", then inside of sudoku_main.cpp, set Color off by `SetColor::toggleColor();` This will toggle the Color off.

Format of SudokuBoard input file. A proper Sudoku input file will have 9 rows each with 9 characters. A space indicates no permanent value at this location; a number will indicate this value at the appropriate location. The first 9 rows represent all permanent values and the next 9 rows (if they exist) represent all user input values. A sample board is:

```
  1  542
2      9
 834
42 98 7 3
  61549
8 9 27 46
     327
 7      4
 942  6
```

Board1: The above file prints.

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

Solution for "Board 1"

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

Testing: test1.cpp

Since testing is critical to the success of writing good code, your testing for all functions must go into file test1.cpp. **Note: it is considered good practice to write the test suite BEFORE you implement a function.** This way, once you have written code for a function, you can test it immediately. It is always good to know if you are correct or not -- without using a submit to the autograder. It is also a very good way to cut coding time by a significant amount.

Pros for testing as you go

- save significant development time
- pinpoint errors
- know immediately where the problem resides -- if there is a problem
- easy to do
- an extremely useful skill
- did we mention **save time**

There are more deals and hints on testing throughout this document and in the RME's of functions. Testing MUST be done with NO input from the keyboard. We suggest implementing and testing functions in the following order:

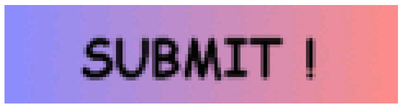
1) void clearInput (used by many of the functions coming up)
2) void openFile
3) SudokuBoard::SudokuBoard (class constructor)
4) SudokuBoard::loadBoard
5) bool getCoords
6) bool SudokuBoard::validMove
7) void SudokuBoard::placeValue
8) void SudokuBoard::deleteValue
9) bool SudokuBoard::isFull
10) void SudokuBoard::playGame

NOTE: Make sure you have that

```
#include "sudoku.h"
#include "SudokuBoard.h"
```

at the top of your test1.cpp. It is needed or your code will not compile. This line is inserting all those prototypes that are in `sudoku.h` and `SudokuBoard.h` into your program. It will make the compiler very happy.☺

Submission Requirements: sudoku.cpp
SudokuBoard.cpp
test1.cpp



test1.cpp
The testing file gets submitted to

Project 5-Test Suite

due at 6:00 but accepted until 11:59:59 pm on the due date. You have 2 submits per day with feedback plus one "wildcard" submit for this portion of Project 5.

SudokuBoard.cpp and sudoku.cpp
These files should be submitted to

Project 5-Sudoku

due at 6:00 but accepted until 11:59:59 pm on the due date. You have 2 submits with feedback per day plus one "wildcard" submit for this portion of Project 5.

Partners - BOTH submit - plus names in program

Although you are welcome to work alone if you wish, we strongly urge you to consider partnering up for P5. If you would like a partner but don't know anyone in the class, we encourage you to post on Piazza's study group forums if you want to find someone.

As a further reminder, a partnership is defined as two people. You are encouraged to help each other and discuss the project, but don't share project code with anyone but your partner.

Both partner's names and uniqnames go in both students' submits, and BOTH must do the final submit. Your code may be identical to your partner's or be different, whatever you want. We will grade them all individually for runnability; partnering means we will ignore any similarities between the two partners. This also means that if only one partner submits only that one will get credit.

We STRONGLY suggest you start your project with the names/uniqnames entered such that you don't forget to do this.

Note: if you begin this project with a partner, you must finish it with the same partner.

Bonus Point Policy

Bonus Point Policy applies to runnability and test suite portions of this project. Bonus Points do NOT apply to the style portion.

Late Day Policy

You are allowed 3 late days for the entire semester with no penalty. The autograder is only a guideline but not exact.

If a 4th late day is used, you will receive 50% of your project score.
If a 5th late day is used, you will receive a 0 on the project. This is non-negotiable.

Keep in mind we take your last submit to determine bonus points and late days. We do not take the highest score. We do not go back and take an earlier submit. We do not excuse the one you didn't mean to submit. We do not excuse a submit because you didn't read carefully enough.

Check ctools/resources/Administrivia/policies for the full statement of "late" policies.

Code Quality:

We will review your functions.cpp file for code quality. This is done by extremely knowledgeable people. For details on what we will be looking for refer to:

ctools/resources/Style Guidelines

Read the style guideline so you will know: How to ACE “style” points

Partner Style grading: If you do this project with a partner, we will randomly choose one of the partner's to grade for style. Both partners will be given the same style score.