1: The Sudoku Project

CSCI 6626 / 4526 Fall 2016

1 The Vision Statement.

We want to model a Sudoku game. The playing board has 9 rows with 9 squares in each row. Each square either has a number (stored there when the puzzle was created) or it is blank. A Sudoku player attempts to fill the empty squares with digits 1–9 so that no row or column has two squares with the same digit. In addition, nine boxes are superimposed on the playing board, each with 3 rows and 3 columns. When the solution is finished, a box must contain 9 different digits.

At any stage in the solution, there are between 0 and 9 possible digits that could be written in a particular square. Writing a digit in one square removes that digit from the list of possibilities for all other squares in the same row, column, or box. We would like our implementation to provide an efficient way to determine whether a player's move is legal or not, and to provide a meaningful error comment if the player tries to make an illegal move. We want the game to tell us when we have won it.

When solving a Sudoku puzzle by hand, people do often see that erase a digit is wrong and erase it. We would like the application to have "undo" and "redo" capabilities. Of course, digits must be erased in the reverse order that they were entered, and the digits that were part of the original puzzle cannot be erased.

Sometimes these puzzles are quite difficult, and even a skilled player does not know what to do next. So he guesses. But before guessing and spoiling the board, he makes a copy of it (photo, Xerox). We want to be able to make a copy of the state of the game and return to that state later.

Finally, there are some intriguing variations on Sudoku that I would like to cover, after the basic game is finished.

2 Analysis.

1a. Find the Objects. The first part of an analysis is to identify and name the classes and the functionality that will be needed. Classes are used to represent objects, sets of objects, and processes. Looking at the vision statement, I see these nouns (classes or primitive types):

- The game or game variant: including shape of the puzzle and rules for solving the puzzle.
- Puzzle or board: the playing surface.
- Square: one component of the board.
- The value of a square: a digit 1...9.
- Row, column, and box (groups of nine squares).
- Possibilities: a set of 1 or more digits between 1 and 9 that are legal to put into a given square.
- Files for save and restore: strings for the file names, streams for the I/O.

Most of these things cannot be represented by primitive values, for example, a puzzle or board is much more than a single integer. These will be modeled by classes; the first step is to name the classes. In the process of developing the program, we may discover the need for additional classes.

The possibilities could be represented several ways: as a string of digits, an array of small integers, or a bit vector. The string of digits would be the bulkiest, a small integer would be the most efficient for space.

2. Find the Functions. Also, I see these action verbs (functions):

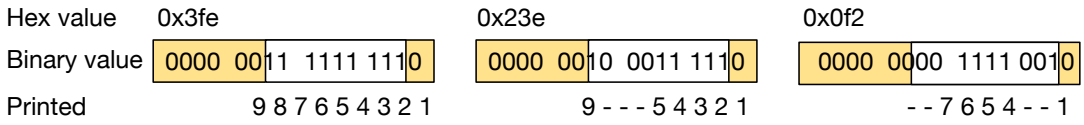
- Input a puzzle or board.
- Fill an empty Square.
- Remove a value from the list of possibilities for neighboring squares.
- Efficiently determine whether a move is legal.
- Provide meaningful error comments.
- Undo a move . . . in reverse order
- Redo a move
- Recognize a value that cannot be erased.
- Save the state of the game to a file.
- Restore the game state from the file.
- Files for save and restore.
- Identify and announce a complete puzzle.

3 Instructions: One Square on a Sudoku Board

You have to start somewhere, and the logical starting place is the lowest-level data structure, the square. Each square has a state and a set of relationships to other squares. This first assignment asks you to implement the State of a square. More parts (data and functions) may be added to this class as the project develops.

Define a class State.

- It should have these data members: value (char), possibilities (short), and fixed (boolean). A value is fixed if it was part of the original puzzle read from the input file.
- A constructor with a char parameter is needed and is the only constructor needed at this time. Initialize the value to the parameter (a dash or a single digit). If the value is a digit, initialize the possibility list to 0 and the fixed flag to true. If it is a '-', initialize the fixed flag to false and the possibility list to 0x3fe (In binary, 0000 0011 1111 1110.) This means that all nine digits are still possible for this square. (A later assignment will change this list of possibilities.)
- Define a default destructor.
- Implement the following function members:
  - move(char ch) : Call say() if this State is fixed, and do not do the move. Otherwise, make ch the new value in the State. The say() function is in tools and is called like fatal but it does not abort.
  - erase(): Call say() if this State is fixed, and do not do the erase. Otherwise, set the value in the State to '-'.
  - A method for `print()` that prints all data in the State in a readable format. Print the possibility list as a series of digits and dashes. For example, if the value was 0x11e, you would print `12345---9`. Do this in a loop by right-shifting the possibility list one bit and masking off the 1's bit, then testing it. Example output is shown below for three different possibility lists.



– Outside the class but inside the .hpp file, declare an inline method for the output operator. It must call your print() function with the appropriate parameter. This definition allows you to output a State as easily as you output an integer or a string.

4 Testing and Submission

Due September 19. Write a main program and a first version of a unit test for this class. Call the unit test from main. In the unit test, call each of the functions defined in the State class and verify that they work properly. Submit evidence that they work. This will test the default possibility lists. To test a different possibility list, change your program (temporarily) to initialize the possibilities to 0x23e or 0x0f2.

5 Advice

This is a very short, very easy program. Don't complicate it. Its purpose is to make sure you understand how to organize and use a project and a class with header and implementation files, and how to do the easiest kind of bit operations. Email for help promptly if you are confused about the instructions or you have any trouble of any sort. Be sure you have something to hand in by the due date. If you finish this early, hand it in early and start on Program 3: Implement the class Square. Instructions will follow.