

SudokuSolver

Generated by Doxygen 1.12.0

1 SudokuSolver	1
2 Class Index	5
2.1 Class List	5
3 File Index	7
3.1 File List	7
4 Class Documentation	9
4.1 CandSet Struct Reference	9
4.1.1 Detailed Description	9
4.1.2 Constructor & Destructor Documentation	10
4.1.2.1 CandSet()	10
4.1.3 Member Function Documentation	10
4.1.3.1 begin()	10
4.1.3.2 cand2str()	10
4.1.3.3 clear()	10
4.1.3.4 end()	10
4.1.3.5 erase()	10
4.1.3.6 insert()	10
4.1.3.7 operator!=()	11
4.1.3.8 operator&&()	11
4.1.3.9 operator+=()	11
4.1.3.10 operator-()	11
4.1.3.11 operator-=()	11
4.1.3.12 operator=()	11
4.1.3.13 operator==(())	11
4.1.3.14 operator" " ()	11
4.1.3.15 remove()	12
4.1.3.16 size()	12
4.1.4 Member Data Documentation	12
4.1.4.1 data	12
4.2 Cell Struct Reference	12
4.2.1 Detailed Description	13
4.2.2 Constructor & Destructor Documentation	13
4.2.2.1 Cell()	13
4.2.3 Member Function Documentation	13
4.2.3.1 cord2str()	13
4.2.3.2 init()	14
4.2.3.3 isEq()	14
4.2.3.4 isGap()	14
4.2.3.5 lc()	14
4.2.4 Member Data Documentation	14

4.2.4.1 blk	14
4.2.4.2 blkidx	14
4.2.4.3 candidates	14
4.2.4.4 col	14
4.2.4.5 colBlkPos	15
4.2.4.6 pairColor	15
4.2.4.7 row	15
4.2.4.8 rowBlkPos	15
4.2.4.9 val	15
4.3 SudokuBoard Class Reference	15
4.3.1 Detailed Description	16
4.3.2 Constructor & Destructor Documentation	16
4.3.2.1 SudokuBoard()	16
4.3.3 Member Function Documentation	16
4.3.3.1 applyStrategies()	16
4.3.3.2 at()	16
4.3.3.3 atBlock()	16
4.3.3.4 checkCellForHiddenPair()	16
4.3.3.5 checkCellForHiddenSingle()	16
4.3.3.6 checkCellForLockedCandsInBlocks()	17
4.3.3.7 checkCellForNakedPair()	17
4.3.3.8 checkCellForNakedSingle()	17
4.3.3.9 checkCellForNakedTriplet()	17
4.3.3.10 checkCellForXWing()	17
4.3.3.11 checkCellForXYWing()	17
4.3.3.12 checkForIntersectingColorPairs()	17
4.3.3.13 collectCands()	18
4.3.3.14 isInBlock()	18
4.3.3.15 isInCol()	18
4.3.3.16 isInRow()	18
4.3.3.17 print()	18
4.3.3.18 printSolvingSteps()	18
4.3.3.19 setFinalValue()	18
4.3.3.20 solve()	18
4.3.3.21 updateCandsInBlock()	19
4.3.3.22 updateCandsInCol()	19
4.3.3.23 updateCandsInRow()	19
4.3.3.24 valid()	19
5 File Documentation	21
5.1 /Users/paulkeydel/Documents/coding projects/SudokuSolver/main.cpp File Reference	21
5.1.1 Function Documentation	21

5.1.1.1 <code>getBoardFromFile()</code>	21
5.1.1.2 <code>getBoardFromStdin()</code>	21
5.1.1.3 <code>main()</code>	22
5.1.1.4 <code>saveBoardToFile()</code>	22
5.2 <code>/Users/paulkeydel/Documents/coding projects/SudokuSolver/README.md</code> File Reference	22
5.3 <code>/Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.cpp</code> File Reference	22
5.4 <code>/Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.h</code> File Reference	22
5.5 <code>/Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.h</code>	23
Index	25

Chapter 1

SudokuSolver

This C++ program can solve Sudokus by applying human strategies. A simple text-based user interface is provided in order to easily enter the quiz. It's also possible to read a quiz from file. A few test boards are in this repo.

How does the solving algorithm work? First, the program passes through all empty cells and collects all possible candidates for each cell. Depending on how many entries are given in the same row, column or block, an empty cell can have one or more possible candidates. Based on logical rules and dependencies between cells, the algorithm then tries to eliminate candidates until only one number is left in the set. This, however, can not always be guaranteed. Sudokus can become very complex and because strategic solving needs to include many constellations and subcases, the candidate reduction is not always successful. Even if this implementation says the quiz is not solvable, you might have the chance to find a solution with pen and paper. So far, this implementation focuses on 9 logical strategies.

- Hidden singles
- Naked singles
- Hidden pairs
- Naked pairs
- Naked triplets
- Locked candidates
- X-wings
- XY-wings
- Colored pairs

For further descriptions and illustrations see [https://www.sudoku9x9.com/sudoku_solving_↵
techniques.php](https://www.sudoku9x9.com/sudoku_solving_techniques.php)

How to use the algorithm? Just compile the project using the makefile (run `make` or `make debug`), execute `./sudoku` in terminal and enter your Sudoku. Please note that the makefile is configured to use the gcc compiler (change if you're on a non-UNIX system).

As an example you might solve the quiz `test.txt` from the `testboard` directory. Then simply run `./sudoku testboards/test.txt`. The program will solve the quiz and prints all solving steps starting with the (row, col)-coordinates of the current cell.

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

```

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

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

```

```

(0, 1): Cands {6} are locked in block
(0, 4): Cands {3, 6} are locked in col
(0, 6): Cands {8} are locked in col
(2, 7): Hidden Single
(2, 8): Naked Triplet in col with cell (7, 8) and cell (8, 8)
(3, 4): Cands {1} are locked in row
(3, 6): Cands {2} are locked in col
(3, 8): Naked Single
(4, 0): Cands {8} are locked in block
(4, 3): row-wise X-Wing with diag cell(5, 7)
(4, 5): Hidden Single
(4, 7): Cands {3} are locked in block
(5, 0): Hidden Single
(5, 3): Hidden Pair {3, 5} with cell (4, 3)
(6, 6): Hidden Single
(7, 1): Hidden Single
(7, 2): Hidden Single
(7, 6): Cands {7} are locked in block
(7, 7): Naked Triplet in block with cell (7, 8) and cell (8, 8)
(7, 8): Naked Pair with cell (8, 8)
(8, 2): Hidden Single
(8, 3): Hidden Pair {4, 7} with cell (1, 3)
(8, 5): Hidden Single
(8, 6): Hidden Single
(8, 8): Naked Pair with cell (7, 8)
(0, 0): Cands {3} are locked in block
(0, 5): Naked Triplet in block with cell (1, 3) and cell (2, 5)
(1, 0): Cands {1, 2, 5} are locked in block
(1, 3): Naked Pair with cell (2, 5)
(1, 4): Naked Triplet in row with cell (1, 6) and cell (1, 7)
(1, 6): Cands {4} are locked in block
(2, 5): Hidden Single
(2, 8): Naked Single
(3, 2): Hidden Single
(3, 4): Hidden Single
(3, 5): Naked Single
(3, 6): Cands {2, 7} are locked in block
(6, 0): Cands {4} are locked in block
(6, 4): Hidden Single
(6, 5): Hidden Single
(7, 4): row-wise X-Wing with diag cell(8, 8)
(7, 6): Naked Single
(7, 7): Naked Single
(7, 8): Hidden Single
(8, 0): Hidden Single
(8, 3): Naked Single
(8, 4): Hidden Single
(8, 8): Naked Single
(0, 5): Naked Single
(1, 3): Naked Single
(2, 0): Naked Single
(2, 1): Hidden Single
(2, 4): Naked Single
(3, 1): Naked Single
(3, 6): Naked Single
(4, 0): Naked Triplet in row with cell (4, 1) and cell (4, 2)
(4, 1): Cands {4, 5, 9} are locked in block
(4, 3): Naked Single
(4, 6): Hidden Single
(4, 7): Naked Single
(5, 1): Naked Single
(5, 2): Naked Single
(5, 3): Naked Single
(5, 6): Naked Single
(5, 7): Naked Single
(7, 4): Naked Single
(0, 0): Naked Single
(0, 1): Hidden Single
(0, 2): Naked Single
(0, 4): Hidden Single
(0, 6): Naked Single
(1, 0): Hidden Single
(1, 1): Naked Single
(1, 2): Naked Single
(1, 4): Naked Single
(1, 6): Naked Single
(1, 7): Naked Single

```

```
(4, 0): Naked Single  
(4, 1): Hidden Single  
(4, 2): Naked Single  
(6, 0): Naked Single  
(6, 1): Naked Single  
(6, 2): Naked Single  
Solved
```

Documentation: The repository additionally contains a Doxygen config file. If you have Doxygen installed, run `make doc` to create the documentation. You'll get a html and a pdf documentation (pdf is in `doc` folder). If you don't have Doxygen, you'll find all relevant comments in [solver.h](#).

Python implementation: An older version of the solving algorithm was written in Python. Use `python3 sudoku.py <file with board>` to test this.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

CandSet	9
Cell	12
SudokuBoard	15

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

/Users/paulkeydel/Documents/coding projects/SudokuSolver/ main.cpp	21
/Users/paulkeydel/Documents/coding projects/SudokuSolver/ solver.cpp	22
/Users/paulkeydel/Documents/coding projects/SudokuSolver/ solver.h	22

Chapter 4

Class Documentation

4.1 CandSet Struct Reference

```
#include <solver.h>
```

Public Member Functions

- [CandSet](#) ()
- void [insert](#) (int dig)
- void [erase](#) (int dig)
- int [size](#) ()
- void [clear](#) ()
- const std::set< int >::iterator [begin](#) () const
- const std::set< int >::iterator [end](#) () const
- bool [remove](#) ([CandSet](#) &set)
- std::string [cand2str](#) ()
- bool [operator==](#) ([CandSet](#) &op)
- bool [operator!=](#) ([CandSet](#) &op)
- [CandSet](#) [operator-](#) ([CandSet](#) &op)
- [CandSet](#) [operator&&](#) ([CandSet](#) &op)
- [CandSet](#) [operator||](#) ([CandSet](#) &op)
- [CandSet](#) & [operator+=](#) ([CandSet](#) &op)
- [CandSet](#) & [operator-=](#) ([CandSet](#) &op)
- [CandSet](#) & [operator=](#) (const [CandSet](#) &op)

Public Attributes

- std::set< int > [data](#)

4.1.1 Detailed Description

[CandSet](#) is used to store and manage all possible candidates of a cell, i.e. the class contains both a container for candidates and several functions to manipulate the set. Manipulating data includes adding new digits, subtracting digits or calculating the union and intersection.

[CandSet::data](#) is a std::set-container for storing all possible candidates. Internally the data structure is an std::set<int>.

To simplify coding, several operators are overloaded in this class. With [CandSet](#) it's possible to use =, ==, !=, -, +=, -=, || and &&. The binary || operator calculates the union between two CandSets while the && operator takes the intersection between the left and right operand. The assignment(=) creates a copy of the source.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 CandSet()

```
CandSet::CandSet () [inline]
```

Creates an empty candidate container. After creation, the objects internal data variable has size 0.

4.1.3 Member Function Documentation

4.1.3.1 begin()

```
const std::set< int >::iterator CandSet::begin () const [inline]
```

4.1.3.2 cand2str()

```
string CandSet::cand2str ()
```

Produces a formatted output string containing the entire set of candidates.

Returns

A string "{cand0, cand1, cand2}".

4.1.3.3 clear()

```
void CandSet::clear () [inline]
```

Reset the object and delete all candidates in set. After this, [CandSet::size](#) is 0.

4.1.3.4 end()

```
const std::set< int >::iterator CandSet::end () const [inline]
```

4.1.3.5 erase()

```
void CandSet::erase (  
    int dig) [inline]
```

Deletes a specific candidate number from list.

Parameters

<i>dig</i>	The digit to be deleted from set (between 1 and 9).
------------	---

4.1.3.6 insert()

```
void CandSet::insert (  
    int dig) [inline]
```

Inserts a specific candidate number into the list.

Parameters

<i>dig</i>	The digit to be inserted into the set (between 1 and 9).
------------	--

4.1.3.7 operator!=()

```
bool CandSet::operator!= (
    CandSet & op) [inline]
```

4.1.3.8 operator&&()

```
CandSet CandSet::operator&& (
    CandSet & op)
```

4.1.3.9 operator+=()

```
CandSet & CandSet::operator+= (
    CandSet & op)
```

4.1.3.10 operator-()

```
CandSet CandSet::operator- (
    CandSet & op)
```

4.1.3.11 operator-=()

```
CandSet & CandSet::operator-= (
    CandSet & op)
```

4.1.3.12 operator=()

```
CandSet & CandSet::operator= (
    const CandSet & op)
```

4.1.3.13 operator==(())

```
bool CandSet::operator==(
    CandSet & op) [inline]
```

4.1.3.14 operator"|"|()

```
CandSet CandSet::operator|| (
    CandSet & op)
```

4.1.3.15 remove()

```
bool CandSet::remove (
    CandSet & set)
```

Removes all candidates that are given by the argument.

Returns

<true> if at least one candidate could successfully be removed. <false> if the size could not be reduced.

4.1.3.16 size()

```
int CandSet::size () [inline]
```

Gives the number of listed candidates.

Returns

dig The number of candidates as an int value. If [CandSet::data](#) is empty, 0 is returned.

4.1.4 Member Data Documentation

4.1.4.1 data

```
std::set<int> CandSet::data
```

The documentation for this struct was generated from the following files:

- [/Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.h](#)
- [/Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.cpp](#)

4.2 Cell Struct Reference

```
#include <solver.h>
```

Public Member Functions

- [Cell](#) ()
- void [init](#) (int idx, int digit)
- std::string [cord2str](#) ()
- bool [isEq](#) (int dig)
- bool [isGap](#) ()
- const int [lc](#) ()

Public Attributes

- int `val`
- int `row`
- int `col`
- int `blk`
- int `blkidx`
- int `rowBlkPos`
- int `colBlkPos`
- int `pairColor`
- `CandSet` `candidates`

4.2.1 Detailed Description

Each of all 81 Sudoku cells are mapped to an object of the class `Cell`. The `Cell` class contains the coordinates, the digit (0 if it's a gap) and the color parameter for the coloring pair algorithm. Here is an overview about the class members:

`Cell::val` is an int and contains the digit ($0 \leq \text{val} \leq 9$, 0 = empty).

`Cell::row` and `Cell::col` are used to index all board cells. The (row, col)-coordinate system starts in the upper left with (row, col) = (0, 0) and ends with (8, 8) in the bottom right.

`Cell::blk` and `Cell::blkidx` form a coordinate system based on the 3x3-subblocks which will be counted in Z-scan order, starting with subblock `blk` = 0 in the upper left. `blkidx` addresses all 9 cells within the 3x3 block by row-wise indexing, $0 \leq \text{blkidx} < 9$.

`Cell::rowBlkPos` and `Cell::colBlkPos` are (row, col)-coordinates referencing to the upper left cell within the current subblock. They are generally multiple of 3 (= 0, 3 or 6).

`Cell::candidates` is used to store and manage all possible candidates. It's an object of class `CandSet`. If `Cell::val` is between 1 and 9, `Cell::candidates` will be an empty set.

`Cell::pairColor` is needed for pairing colors. All identical candidate pairs in board can be colored by setting `Cell::pairColor` alternately to 0 and 1.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 `Cell()`

```
Cell::Cell () [inline]
```

4.2.3 Member Function Documentation

4.2.3.1 `cord2str()`

```
string Cell::cord2str ()
```

Get current coordinates as formatted string.

4.2.3.2 init()

```
void Cell::init (
    int idx,
    int digit)
```

Set all class parameters (position and digit) based on the Z-ordered cell index.

4.2.3.3 isEq()

```
bool Cell::isEq (
    int dig) [inline]
```

Is equal to [Cell::val](#)?

4.2.3.4 isGap()

```
bool Cell::isGap () [inline]
```

Is cell unknown?

4.2.3.5 lc()

```
const int Cell::lc () [inline]
```

Get length of cand's set.

4.2.4 Member Data Documentation

4.2.4.1 blk

```
int Cell::blk
```

4.2.4.2 blkidx

```
int Cell::blkidx
```

4.2.4.3 candidates

```
CandSet Cell::candidates
```

4.2.4.4 col

```
int Cell::col
```

4.2.4.5 colBlkPos

```
int Cell::colBlkPos
```

4.2.4.6 pairColor

```
int Cell::pairColor
```

4.2.4.7 row

```
int Cell::row
```

4.2.4.8 rowBlkPos

```
int Cell::rowBlkPos
```

4.2.4.9 val

```
int Cell::val
```

The documentation for this struct was generated from the following files:

- /Users/paulkeydel/Documents/coding projects/SudokuSolver/[solver.h](#)
- /Users/paulkeydel/Documents/coding projects/SudokuSolver/[solver.cpp](#)

4.3 SudokuBoard Class Reference

```
#include <solver.h>
```

Public Member Functions

- [SudokuBoard](#) (int *board)
- void [print](#) ()
- void [printSolvingSteps](#) ()
- [Cell](#) & [at](#) (int row, int col)
- [Cell](#) & [atBlock](#) (int block, int index)
- bool [isInCol](#) (int col, int digit)
- bool [isInRow](#) (int row, int digit)
- bool [isInBlock](#) (int block, int digit)
- bool [valid](#) ()
- void [collectCands](#) ()
- bool [updateCandsInRow](#) (int row, std::vector< int > excludedPositions, [CandSet](#) digits)
- bool [updateCandsInCol](#) (int col, std::vector< int > excludedPositions, [CandSet](#) digits)
- bool [updateCandsInBlock](#) (int blk, std::vector< int > excludedPositions, [CandSet](#) digits)
- void [setFinalValue](#) (int row, int col)
- bool [checkCellForNakedSingle](#) (int row, int col)
- bool [checkCellForHiddenSingle](#) (int row, int col)
- bool [checkCellForNakedPair](#) (int row, int col)
- bool [checkCellForHiddenPair](#) (int row, int col)
- bool [checkCellForNakedTriplet](#) (int row, int col)
- bool [checkCellForXWing](#) (int row, int col)
- bool [checkCellForXYWing](#) (int row, int col)
- bool [checkCellForLockedCandsInBlocks](#) (int row, int col)
- void [checkForIntersectingColorPairs](#) (int row, int col, int row1=-1, int col1=-1, int color=0)
- void [applyStrategies](#) ()
- bool [solve](#) (int numIterations=INT_MAX)

4.3.1 Detailed Description

[SudokuBoard](#) represents the whole board and comprises all 81 cells of type [Cell](#). The class additionally includes methods for collecting and updating candidates as well as solving techniques.

Use [SudokuBoard::solve](#) to find the solution of the current quiz. The algorithm iteratively applies all implemented techniques to each cell. The methods [SudokuBoard::print](#) and [SudokuBoard::printSolvingSteps](#) print the resulting board and all effective solving steps.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 SudokuBoard()

```
SudokuBoard::SudokuBoard (  
    int * board)
```

4.3.3 Member Function Documentation

4.3.3.1 applyStrategies()

```
void SudokuBoard::applyStrategies ()
```

4.3.3.2 at()

```
Cell & SudokuBoard::at (  
    int row,  
    int col)
```

4.3.3.3 atBlock()

```
Cell & SudokuBoard::atBlock (  
    int block,  
    int index)
```

4.3.3.4 checkCellForHiddenPair()

```
bool SudokuBoard::checkCellForHiddenPair (  
    int row,  
    int col)
```

4.3.3.5 checkCellForHiddenSingle()

```
bool SudokuBoard::checkCellForHiddenSingle (  
    int row,  
    int col)
```

4.3.3.6 checkCellForLockedCandsInBlocks()

```
bool SudokuBoard::checkCellForLockedCandsInBlocks (
    int row,
    int col)
```

4.3.3.7 checkCellForNakedPair()

```
bool SudokuBoard::checkCellForNakedPair (
    int row,
    int col)
```

4.3.3.8 checkCellForNakedSingle()

```
bool SudokuBoard::checkCellForNakedSingle (
    int row,
    int col)
```

4.3.3.9 checkCellForNakedTriplet()

```
bool SudokuBoard::checkCellForNakedTriplet (
    int row,
    int col)
```

4.3.3.10 checkCellForXWing()

```
bool SudokuBoard::checkCellForXWing (
    int row,
    int col)
```

4.3.3.11 checkCellForXYWing()

```
bool SudokuBoard::checkCellForXYWing (
    int row,
    int col)
```

4.3.3.12 checkForIntersectingColorPairs()

```
void SudokuBoard::checkForIntersectingColorPairs (
    int row,
    int col,
    int row1 = -1,
    int col1 = -1,
    int color = 0)
```

4.3.3.13 collectCands()

```
void SudokuBoard::collectCands ()
```

4.3.3.14 isInBlock()

```
bool SudokuBoard::isInBlock (
    int block,
    int digit)
```

4.3.3.15 isInCol()

```
bool SudokuBoard::isInCol (
    int col,
    int digit)
```

4.3.3.16 isInRow()

```
bool SudokuBoard::isInRow (
    int row,
    int digit)
```

4.3.3.17 print()

```
void SudokuBoard::print ()
```

4.3.3.18 printSolvingSteps()

```
void SudokuBoard::printSolvingSteps ()
```

4.3.3.19 setFinalValue()

```
void SudokuBoard::setFinalValue (
    int row,
    int col)
```

4.3.3.20 solve()

```
bool SudokuBoard::solve (
    int numIterations = INT_MAX)
```


4.3.3.21 updateCandsInBlock()

```
bool SudokuBoard::updateCandsInBlock (
    int blk,
    std::vector< int > excludedPositions,
    CandSet digits)
```

4.3.3.22 updateCandsInCol()

```
bool SudokuBoard::updateCandsInCol (
    int col,
    std::vector< int > excludedPositions,
    CandSet digits)
```

4.3.3.23 updateCandsInRow()

```
bool SudokuBoard::updateCandsInRow (
    int row,
    std::vector< int > excludedPositions,
    CandSet digits)
```

4.3.3.24 valid()

```
bool SudokuBoard::valid ()
```

The documentation for this class was generated from the following files:

- /Users/paulkeydel/Documents/coding projects/SudokuSolver/[solver.h](#)
- /Users/paulkeydel/Documents/coding projects/SudokuSolver/[solver.cpp](#)

Chapter 5

File Documentation

5.1 /Users/paulkeydel/Documents/coding projects/SudokuSolver/main.cpp File Reference

```
#include <curses.h>
#include <vector>
#include <cassert>
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include "solver.h"
```

Functions

- void [getBoardFromStdin](#) (int *board)
- void [getBoardFromFile](#) (std::string fname, int *board)
- void [saveBoardToFile](#) (std::string fname, int *board)
- int [main](#) (int argc, char *argv[])

5.1.1 Function Documentation

5.1.1.1 [getBoardFromFile\(\)](#)

```
void getBoardFromFile (
    std::string fname,
    int * board)
```

5.1.1.2 [getBoardFromStdin\(\)](#)

```
void getBoardFromStdin (
    int * board)
```

5.1.1.3 main()

```
int main (
    int argc,
    char * argv[])
```

5.1.1.4 saveBoardToFile()

```
void saveBoardToFile (
    std::string fname,
    int * board)
```

5.2 /Users/paulkeydel/Documents/coding projects/SudokuSolver/README.md File Reference

5.3 /Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.cpp File Reference

```
#include "solver.h"
#include <cassert>
#include <iostream>
```

5.4 /Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.h File Reference

```
#include <set>
#include <array>
#include <vector>
```

Classes

- struct [CandSet](#)
- struct [Cell](#)
- class [SudokuBoard](#)

5.5 /Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.h

[Go to the documentation of this file.](#)

```
00001 #include <set>
00002 #include <array>
00003 #include <vector>
00004
00012 struct CandSet
00013 {
00014     std::set<int> data;
00016     CandSet() {};
00020     void insert(int dig) { this->data.insert(dig); }
00024     void erase(int dig) { this->data.erase(dig); }
00028     int size() { return (int)(this->data.size()); }
00031     void clear() { this->data.clear(); }
00032     const std::set<int>::iterator begin() const { return this->data.begin(); }
00033     const std::set<int>::iterator end() const { return this->data.end(); }
00037     bool remove(CandSet& set);
00041     std::string cand2str();
00042     bool operator==(CandSet& op) { return this->data == op.data; }
00043     bool operator!=(CandSet& op) { return this->data != op.data; }
00044     CandSet operator-(CandSet& op);
00045     CandSet operator&&(CandSet& op);
00046     CandSet operator|| (CandSet& op);
00047     CandSet& operator+=(CandSet& op);
00048     CandSet& operator-=(CandSet& op);
00049     CandSet& operator=(const CandSet& op);
00050 };
00051
00067 struct Cell
00068 {
00069     //digit of the cell
00070     int val;
00071     //position parameters
00072     int row;
00073     int col;
00074     int blk;
00075     int blkidx;
00076     //upper left cell in current subblock
00077     int rowBlkPos;
00078     int colBlkPos;
00079     //color for color pair algorithm
00080     int pairColor;
00081     //set for storing candidates
00082     CandSet candidates;
00083     //methods
00084     Cell() {};
00086     void init(int idx, int digit);
00088     std::string cord2str();
00090     bool isEq(int dig) { return (this->val == dig); }
00092     bool isGap() { return (this->val == 0); }
00094     const int lc() { return (int)this->candidates.data.size(); }
00095 };
00096
00102 class SudokuBoard
00103 {
00104 private:
00105     std::array<Cell, 81> b;
00106     //stuff for list of solving steps
00107     std::vector<std::pair<int, std::string>> solvingSteps;
00108     void appendSolvStep(int row, int col, std::string text, bool bReducedCands);
00109 public:
00110     SudokuBoard(int* board);
00111     void print();
00112     void printSolvingSteps();
00113     Cell& at(int row, int col);
00114     Cell& atBlock(int block, int index);
00115     bool isInCol(int col, int digit);
00116     bool isInRow(int row, int digit);
00117     bool isInBlock(int block, int digit);
00118     bool valid();
00119     //methods for managing candidate list
00120     void collectCands();
00121     bool updateCandsInRow(int row, std::vector<int> excludedPositions, CandSet digits);
00122     bool updateCandsInCol(int col, std::vector<int> excludedPositions, CandSet digits);
00123     bool updateCandsInBlock(int blk, std::vector<int> excludedPositions, CandSet digits);
00124     void setFinalValue(int row, int col);
00125     //solving techniques
00126     bool checkCellForNakedSingle(int row, int col);
00127     bool checkCellForHiddenSingle(int row, int col);
00128     bool checkCellForNakedPair(int row, int col);
00129     bool checkCellForHiddenPair(int row, int col);
00130     bool checkCellForNakedTriplet(int row, int col);
```

```
00131     bool checkCellForXWing(int row, int col);
00132     bool checkCellForXYWing(int row, int col);
00133     bool checkCellForLockedCandsInBlocks(int row, int col);
00134     void checkForIntersectingColorPairs(int row, int col, int row1 = -1, int col1 = -1, int color =
00135         0);
00135     void applyStrategies();
00136     bool solve(int numIterations = INT_MAX);
00137 };
```

Index

[/Users/paulkeydel/Documents/coding projects/SudokuSolver/README.md](#),
[22](#)

[/Users/paulkeydel/Documents/coding projects/SudokuSolver/main.py](#),
[21](#)

[/Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.py](#),
[22](#)

[/Users/paulkeydel/Documents/coding projects/SudokuSolver/solver2.py](#),
[22](#)

applyStrategies
 SudokuBoard, [16](#)

at
 SudokuBoard, [16](#)

atBlock
 SudokuBoard, [16](#)

begin
 CandSet, [10](#)

blk
 Cell, [14](#)

blkidx
 Cell, [14](#)

cand2str
 CandSet, [10](#)

candidates
 Cell, [14](#)

CandSet, [9](#)
 begin, [10](#)
 cand2str, [10](#)
 CandSet, [10](#)
 clear, [10](#)
 data, [12](#)
 end, [10](#)
 erase, [10](#)
 insert, [10](#)
 operator!=, [11](#)
 operator+=, [11](#)
 operator-, [11](#)
 operator-=, [11](#)
 operator=, [11](#)
 operator==, [11](#)
 operator&&, [11](#)
 operator | |, [11](#)
 remove, [11](#)
 size, [12](#)

Cell, [12](#)
 blk, [14](#)
 blkidx, [14](#)
 candidates, [14](#)
 col, [14](#)
 colBlkPos, [14](#)
 cord2str, [13](#)
 isEq, [14](#)
 isGap, [14](#)
 lc, [14](#)
 pairColor, [15](#)
 row, [15](#)
 rowBlkPos, [15](#)
 val, [15](#)

checkCellForHiddenPair
 SudokuBoard, [16](#)

checkCellForHiddenSingle
 SudokuBoard, [16](#)

checkCellForLockedCandsInBlocks
 SudokuBoard, [16](#)

checkCellForNakedPair
 SudokuBoard, [17](#)

checkCellForNakedSingle
 SudokuBoard, [17](#)

checkCellForNakedTriplet
 SudokuBoard, [17](#)

checkCellForXWing
 SudokuBoard, [17](#)

checkCellForXYWing
 SudokuBoard, [17](#)

checkForIntersectingColorPairs
 SudokuBoard, [17](#)

clear
 CandSet, [10](#)

col
 Cell, [14](#)

colBlkPos
 Cell, [14](#)

collectCands
 SudokuBoard, [17](#)

cord2str
 Cell, [13](#)

data
 CandSet, [12](#)

end
 CandSet, [10](#)

erase
 CandSet, [10](#)

getBoardFromFile

- main.cpp, 21
- getBoardFromStdin
 - main.cpp, 21
- init
 - Cell, 13
- insert
 - CandSet, 10
- isEq
 - Cell, 14
- isGap
 - Cell, 14
- isInBlock
 - SudokuBoard, 18
- isInCol
 - SudokuBoard, 18
- isInRow
 - SudokuBoard, 18
- lc
 - Cell, 14
- main
 - main.cpp, 21
- main.cpp
 - getBoardFromFile, 21
 - getBoardFromStdin, 21
 - main, 21
 - saveBoardToFile, 22
- operator!=
 - CandSet, 11
- operator+=
 - CandSet, 11
- operator-
 - CandSet, 11
- operator-=
 - CandSet, 11
- operator=
 - CandSet, 11
- operator==
 - CandSet, 11
- operator&&
 - CandSet, 11
- operator | |
 - CandSet, 11
- pairColor
 - Cell, 15
- print
 - SudokuBoard, 18
- printSolvingSteps
 - SudokuBoard, 18
- remove
 - CandSet, 11
- row
 - Cell, 15
- rowBlkPos
 - Cell, 15
- saveBoardToFile
 - main.cpp, 22
- setFinalValue
 - SudokuBoard, 18
- size
 - CandSet, 12
- solve
 - SudokuBoard, 18
- SudokuBoard, 15
 - applyStrategies, 16
 - at, 16
 - atBlock, 16
 - checkCellForHiddenPair, 16
 - checkCellForHiddenSingle, 16
 - checkCellForLockedCandsInBlocks, 16
 - checkCellForNakedPair, 17
 - checkCellForNakedSingle, 17
 - checkCellForNakedTriplet, 17
 - checkCellForXWing, 17
 - checkCellForXYWing, 17
 - checkForIntersectingColorPairs, 17
 - collectCands, 17
 - isInBlock, 18
 - isInCol, 18
 - isInRow, 18
 - print, 18
 - printSolvingSteps, 18
 - setFinalValue, 18
 - solve, 18
 - SudokuBoard, 16
 - updateCandsInBlock, 18
 - updateCandsInCol, 19
 - updateCandsInRow, 19
 - valid, 19
- SudokuSolver, 1
- updateCandsInBlock
 - SudokuBoard, 18
- updateCandsInCol
 - SudokuBoard, 19
- updateCandsInRow
 - SudokuBoard, 19
- val
 - Cell, 15
- valid
 - SudokuBoard, 19