

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 appendSolvStep()	16
4.3.3.2 applyStrategies()	17
4.3.3.3 at()	17
4.3.3.4 atBlock()	17
4.3.3.5 checkCellForHiddenPair()	17
4.3.3.6 checkCellForHiddenSingle()	17
4.3.3.7 checkCellForLockedCandsInBlocks()	17
4.3.3.8 checkCellForNakedPair()	17
4.3.3.9 checkCellForNakedSingle()	17
4.3.3.10 checkCellForNakedTriplet()	18
4.3.3.11 checkCellForXWing()	18
4.3.3.12 checkCellForXYWing()	18
4.3.3.13 checkForIntersectingColorPairs()	18
4.3.3.14 collectCands()	18
4.3.3.15 isInBlock()	18
4.3.3.16 isInCol()	18
4.3.3.17 isInRow()	19
4.3.3.18 print()	19
4.3.3.19 printSolvingSteps()	19
4.3.3.20 setFinalValue()	19
4.3.3.21 solve()	19
4.3.3.22 updateCandsInBlock()	19
4.3.3.23 updateCandsInCol()	19
4.3.3.24 updateCandsInRow()	19
4.3.3.25 valid()	20
4.3.4 Member Data Documentation	20
4.3.4.1 b	20
4.3.4.2 latexCode	20

4.3.4.3 solvingSteps	20
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 createPdfSolvSteps()	21
5.1.1.2 getBoardFromFile()	21
5.1.1.3 getBoardFromStdin()	22
5.1.1.4 main()	22
5.1.1.5 saveBoardToFile()	22
5.2 /Users/paulkeydel/Documents/coding projects/SudokuSolver/README.md File Reference	22
5.3 /Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.cpp File Reference	22
5.4 /Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.h File Reference	22
5.5 /Users/paulkeydel/Documents/coding projects/SudokuSolver/solver.h	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 both [https://www.sudoku9x9.com/sudoku_solving←_techniques.php](https://www.sudoku9x9.com/sudoku_solving_techniques.php) and my own [visualizations](#).

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)

Private 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 an internal container for storing all possible candidates. It's of type `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 [CandSet](#)s 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 [private]
```

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](#) ()
- std::string & [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)
- bool [checkForIntersectingColorPairs](#) (int row, int col, int row1=-1, int col1=-1, int color=0)
- void [applyStrategies](#) ()
- bool [solve](#) (int numIterations=INT_MAX)

Private Member Functions

- void [appendSolvStep](#) (int row, int col, std::string text, bool bReducedCands)

Private Attributes

- std::array< [Cell](#), 81 > [b](#)
- std::vector< std::pair< int, std::string > > [solvingSteps](#)
- std::string [latexCode](#)

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.

The first step of solving is to collect all candidates in empty cells. For this, [SudokuBoard::solve](#) calls [SudokuBoard::collectCands](#). For each empty cell [SudokuBoard::collectCands](#) checks if a digit between 1 and 9 is missing in current row, column and block by calling helper functions [SudokuBoard::isInRow](#), [SudokuBoard::isInCol](#), [SudokuBoard::isInBlock](#).

The next step is the strategic solving. Here the approach of the implemented techniques depends on whether we have only one or more candidates in the list. If there is a naked single or a hidden single, we'll obtain a candidate set with only one entry. Thus, [Cell::val](#) can directly be set and the found digit can be removed from all candidate sets in same row, column and subblock. The process of setting [Cell::val](#) and updating the corresponding row, column or block is condensed in [SudokuBoard::setFinalValue](#). All other solving techniques are used to piecewise reduce the [CandSet](#) by considering logic patterns and dependencies. Elimination either covers the own [CandSet](#) or the sets in neighborhood. In order to check if an elimination was successful in a row, column or block sub-structure, the algorithm utilises [SudokuBoard::updateCandsInRow](#), [SudokuBoard::updateCandsInCol](#) and [SudokuBoard::updateCandsInBlock](#). Only if all existing candidate sets can be reduced to size 1 the Sudoku solver can give a complete solution.

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 appendSolvStep()

```
void SudokuBoard::appendSolvStep (
    int row,
    int col,
    std::string text,
    bool bReducedCands) [private]
```

4.3.3.2 applyStrategies()

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

4.3.3.3 at()

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

4.3.3.4 atBlock()

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

4.3.3.5 checkCellForHiddenPair()

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

4.3.3.6 checkCellForHiddenSingle()

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

4.3.3.7 checkCellForLockedCandsInBlocks()

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

4.3.3.8 checkCellForNakedPair()

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

4.3.3.9 checkCellForNakedSingle()

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

4.3.3.10 checkCellForNakedTriplet()

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

4.3.3.11 checkCellForXWing()

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

4.3.3.12 checkCellForXYWing()

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

4.3.3.13 checkForIntersectingColorPairs()

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

4.3.3.14 collectCands()

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

4.3.3.15 isInBlock()

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

4.3.3.16 isInCol()

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


4.3.3.17 isInRow()

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

4.3.3.18 print()

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

4.3.3.19 printSolvingSteps()

```
string & SudokuBoard::printSolvingSteps ()
```

4.3.3.20 setFinalValue()

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

4.3.3.21 solve()

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

4.3.3.22 updateCandsInBlock()

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

4.3.3.23 updateCandsInCol()

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

4.3.3.24 updateCandsInRow()

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

4.3.3.25 valid()

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

4.3.4 Member Data Documentation

4.3.4.1 b

```
std::array<Cell, 81> SudokuBoard::b [private]
```

4.3.4.2 latexCode

```
std::string SudokuBoard::latexCode [private]
```

4.3.4.3 solvingSteps

```
std::vector<std::pair<int, std::string> > SudokuBoard::solvingSteps [private]
```

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)
- void [createPdfSolvSteps](#) (std::string &srcCode)
- int [main](#) (int argc, char *argv[])

5.1.1 Function Documentation

5.1.1.1 [createPdfSolvSteps\(\)](#)

```
void createPdfSolvSteps (
    std::string & srcCode)
```

5.1.1.2 [getBoardFromFile\(\)](#)

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

5.1.1.3 getBoardFromStdin()

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

5.1.1.4 main()

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

5.1.1.5 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 private:
00015     std::set<int> data;
00016 public:
00018     CandSet() {};
00022     void insert(int dig) { this->data.insert(dig); }
00026     void erase(int dig) { this->data.erase(dig); }
00030     int size() { return (int)(this->data.size()); }
00033     void clear() { this->data.clear(); }
00034     const std::set<int>::iterator begin() const { return this->data.begin(); }
00035     const std::set<int>::iterator end() const { return this->data.end(); }
00039     bool remove(CandSet& set);
00043     std::string cand2str();
00044     bool operator==(CandSet& op) { return this->data == op.data; }
00045     bool operator!=(CandSet& op) { return this->data != op.data; }
00046     CandSet operator-(CandSet& op);
00047     CandSet operator&&(CandSet& op);
00048     CandSet operator|| (CandSet& op);
00049     CandSet& operator+=(CandSet& op);
00050     CandSet& operator-=(CandSet& op);
00051     CandSet& operator=(const CandSet& op);
00052 };
00053
00069 struct Cell
00070 {
00071     //digit of the cell
00072     int val;
00073     //position parameters
00074     int row;
00075     int col;
00076     int blk;
00077     int blkidx;
00078     //upper left cell in current subblock
00079     int rowBlkPos;
00080     int colBlkPos;
00081     //color for color pair algorithm
00082     int pairColor;
00083     //set for storing candidates
00084     CandSet candidates;
00085     //methods
00086     Cell() {};
00088     void init(int idx, int digit);
00090     std::string cord2str();
00092     bool isEq(int dig) { return (this->val == dig); }
00094     bool isGap() { return (this->val == 0); }
00096     const int lc() { return this->candidates.size(); }
00097 };
00098
00109 class SudokuBoard
00110 {
00111 private:
00112     std::array<Cell, 81> b;
00113     //stuff for list of solving steps
00114     std::vector<std::pair<int, std::string>> solvingSteps;
00115     std::string latexCode;
00116     void appendSolvStep(int row, int col, std::string text, bool bReducedCands);
00117 public:
00118     SudokuBoard(int* board);
00119     void print();
00120     std::string& printSolvingSteps();
00121     Cell& at(int row, int col);
00122     Cell& atBlock(int block, int index);
00123     bool isInCol(int col, int digit);
00124     bool isInRow(int row, int digit);
00125     bool isInBlock(int block, int digit);
00126     bool valid();
00127     //methods for managing candidate list
00128     void collectCands();
00129     bool updateCandsInRow(int row, std::vector<int> excludedPositions, CandSet digits);
00130     bool updateCandsInCol(int col, std::vector<int> excludedPositions, CandSet digits);
00131     bool updateCandsInBlock(int blk, std::vector<int> excludedPositions, CandSet digits);
00132     void setFinalValue(int row, int col);
00133     //solving techniques
00134     bool checkCellForNakedSingle(int row, int col);
00135     bool checkCellForHiddenSingle(int row, int col);

```

```
00136     bool checkCellForNakedPair(int row, int col);
00137     bool checkCellForHiddenPair(int row, int col);
00138     bool checkCellForNakedTriplet(int row, int col);
00139     bool checkCellForXWing(int row, int col);
00140     bool checkCellForXYWing(int row, int col);
00141     bool checkCellForLockedCandsInBlocks(int row, int col);
00142     bool checkForIntersectingColorPairs(int row, int col, int row1 = -1, int col1 = -1, int color =
00143         0);
00143     void applyStrategies();
00144     bool solve(int numIterations = INT_MAX);
00145 };
```

Index

[/Users/paulkeydel/Documents/coding projects/SudokuSolver/README.md](#),
[22](#)
[/Users/paulkeydel/Documents/coding projects/SudokuSolver/main.cpp](#),
[21](#)
[/Users/paulkeydel/Documents/coding projects/SudokuSolver/solve.cpp](#),
[22](#)
[/Users/paulkeydel/Documents/coding projects/SudokuSolver/solveBlkPos](#),
[22](#)

appendSolvStep
 SudokuBoard, [16](#)
applyStrategies
 SudokuBoard, [16](#)
at
 SudokuBoard, [17](#)
atBlock
 SudokuBoard, [17](#)

b
 SudokuBoard, [20](#)
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](#)

checkCellForHiddenPair
 SudokuBoard, [17](#)
checkCellForHiddenSingle
 SudokuBoard, [17](#)
checkCellForLockedCandsInBlocks
 SudokuBoard, [17](#)
checkCellForNakedPair
 SudokuBoard, [17](#)
checkCellForNakedSingle
 SudokuBoard, [17](#)
checkCellForNakedTriplet
 SudokuBoard, [17](#)
checkCellForXWing
 SudokuBoard, [18](#)
checkCellForXYWing
 SudokuBoard, [18](#)
checkForIntersectingColorPairs
 SudokuBoard, [18](#)
clear
 CandSet, [10](#)
col
 Cell, [14](#)
colBlkPos
 Cell, [14](#)
collectCands
 SudokuBoard, [18](#)
cord2str
 Cell, [13](#)
createPdfSolvSteps
 main.cpp, [21](#)

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
- latexCode
 - SudokuBoard, 20
- lc
 - Cell, 14
- main
 - main.cpp, 22
- main.cpp
 - createPdfSolvSteps, 21
 - getBoardFromFile, 21
 - getBoardFromStdin, 21
 - main, 22
 - 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, 19
- printSolvingSteps
 - SudokuBoard, 19
- remove
 - CandSet, 11
- row
 - Cell, 15
- rowBlkPos
 - Cell, 15
- saveBoardToFile
 - main.cpp, 22
- setFinalValue
 - SudokuBoard, 19
- size
 - CandSet, 12
- solve
 - SudokuBoard, 19
- solvingSteps
 - SudokuBoard, 20
- SudokuBoard, 15
 - appendSolvStep, 16
 - applyStrategies, 16
 - at, 17
 - atBlock, 17
 - b, 20
 - checkCellForHiddenPair, 17
 - checkCellForHiddenSingle, 17
 - checkCellForLockedCandsInBlocks, 17
 - checkCellForNakedPair, 17
 - checkCellForNakedSingle, 17
 - checkCellForNakedTriplet, 17
 - checkCellForXWing, 18
 - checkCellForXYWing, 18
 - checkForIntersectingColorPairs, 18
 - collectCands, 18
 - isInBlock, 18
 - isInCol, 18
 - isInRow, 18
 - latexCode, 20
 - print, 19
 - printSolvingSteps, 19
 - setFinalValue, 19
 - solve, 19
 - solvingSteps, 20
 - SudokuBoard, 16
 - updateCandsInBlock, 19
 - updateCandsInCol, 19
 - updateCandsInRow, 19
 - valid, 19
- SudokuSolver, 1
 - updateCandsInBlock
 - SudokuBoard, 19
 - updateCandsInCol
 - SudokuBoard, 19
 - updateCandsInRow
 - SudokuBoard, 19

val

Cell, [15](#)

valid

SudokuBoard, [19](#)