

## Sudoku Project

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>File Index</b>	<b>1</b>
1.1	File List . . . . .	1
<b>2</b>	<b>File Documentation</b>	<b>3</b>
2.1	Display.c File Reference . . . . .	3
2.1.1	Function Documentation . . . . .	4
2.1.1.1	AskSize(int SizeOfPuzzle, int puzzle) . . . . .	4
2.1.1.2	Back(int puzzle, int copy, int SizeOfPuzzle) . . . . .	4
2.1.1.3	ChoseOptions() . . . . .	5
2.1.1.4	GenerateRandomPuzzle(int puzzle, int copy, FILE *nameOfFile, int SizeOfPuzzle) . . . . .	5
2.1.1.5	InsertNumberInPozitionChoice(int puzzle, int copy, int SizeOfPuzzle) . . . . .	6
2.1.1.6	Intro() . . . . .	6
2.1.1.7	SolvePuzzle(int puzzle, FILE *fileToPrint, int SizeOfPuzzle) . . . . .	7
2.2	essentialFunctions.c File Reference . . . . .	7
2.2.1	Function Documentation . . . . .	8
2.2.1.1	Ask() . . . . .	8
2.2.1.2	boxLength(int SizeOfPuzzle) . . . . .	9
2.2.1.3	copyPuzzle(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int copy[SizeOfPuzzle][SizeOfPuzzle]) . . . . .	9
2.2.1.4	counter(int puzzle[SizeOfPuzzle][SizeOfPuzzle]) . . . . .	10
2.2.1.5	insertNumberInPozition(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int SizeOfPuzzle) . . . . .	10
2.2.1.6	OpenFiles(FILE *file) . . . . .	11
2.2.1.7	pastePuzzle(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int copy[SizeOfPuzzle][SizeOfPuzzle]) . . . . .	11
2.2.1.8	printPuzzle(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int SizeOfPuzzle) . . . . .	12

2.2.1.9	printPuzzleInFile(FILE *filename, int puzzle[SizeOfPuzzle][SizeOfPuzzle], int SizeOfPuzzle) . . . . .	12
2.3	GenerateSudokuPuzzle.c File Reference . . . . .	13
2.3.1	Function Documentation . . . . .	14
2.3.1.1	existsBox(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int boxRow, int boxCol, int value) . . . . .	14
2.3.1.2	existsCol(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int col, int value) . . . . .	14
2.3.1.3	existsRow(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int value) . . . . .	15
2.3.1.4	legalAssign(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int col, int value) . . . . .	15
2.3.1.5	randomGeneration(int puzzle[SizeOfPuzzle][SizeOfPuzzle]) . . . . .	16
2.4	main.c File Reference . . . . .	17
2.4.1	Function Documentation . . . . .	18
2.4.1.1	main() . . . . .	18
2.5	sudoku.h File Reference . . . . .	18
2.5.1	Macro Definition Documentation . . . . .	19
2.5.1.1	UNASSIGNED . . . . .	19
2.5.2	Variable Documentation . . . . .	19
2.5.2.1	choice . . . . .	19
2.5.2.2	GameOver . . . . .	19
2.5.2.3	iterations . . . . .	19
2.5.2.4	numberOfSolutions . . . . .	19
2.5.2.5	SizeOfPuzzle . . . . .	20
2.5.2.6	solution . . . . .	20
2.5.2.7	Start . . . . .	20
2.6	SudokuSolver.c File Reference . . . . .	20
2.6.1	Function Documentation . . . . .	21
2.6.1.1	FindUnassigned(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int *row, int *col) . . . . .	21
2.6.1.2	isSafe(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int col, int num) . . . . .	21
2.6.1.3	SolveSudoku(int puzzle[SizeOfPuzzle][SizeOfPuzzle], FILE *fileName) . . . . .	21
2.6.1.4	UsedInBox(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int BoxStartRow, int BoxStartCol, int num) . . . . .	22
2.6.1.5	UsedInCol(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int col, int num) . . . . .	22
2.6.1.6	UsedInRow(int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int num) . . . . .	22
2.6.2	Variable Documentation . . . . .	23
2.6.2.1	solution . . . . .	23

# Chapter 1

## File Index

### 1.1 File List

Here is a list of all files with brief descriptions:

<b>Display.c</b>	3
<b>essentialFunctions.c</b>	7
<b>GenerateSudokuPuzzle.c</b>	13
<b>main.c</b>	17
<b>sudoku.h</b>	18
<b>SudokuSolver.c</b>	20



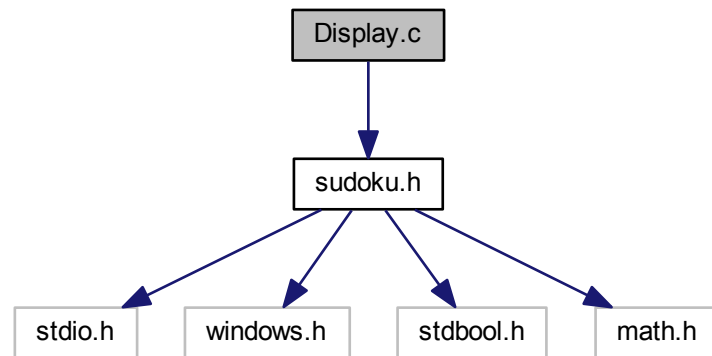
## Chapter 2

# File Documentation

### 2.1 Display.c File Reference

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

Include dependency graph for Display.c:



### Functions

- void **Intro** ()  
*Generate intro interface.*
- void **ChoseOptions** ()  
*Generate chose options interface for switch .*
- void **Back** (int puzzle, int copy, int **SizeOfPuzzle**)  
*Call back-up of last puzzle.*
- void **InsertNumberInPozitionChoice** (int puzzle, int copy, int **SizeOfPuzzle**)  
*Insert number in poztion at your choice !*
- void **GenerateRandomPuzzle** (int puzzle, int copy, FILE \*nameOfFile, int **SizeOfPuzzle**)  
*Generate puzzle.*
- void **SolvePuzzle** (int puzzle, FILE \*fileToPrint, int **SizeOfPuzzle**)  
*A function which will call solving parts to solve the puzzle !*
- void **AskSize** (int **SizeOfPuzzle**, int puzzle)  
*Ask for size of puzzle .*

## 2.1.1 Function Documentation

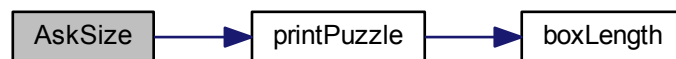
### 2.1.1.1 void AskSize ( int *SizeOfPuzzle*, int *puzzle* )

Ask for size of puzzle .

#### Parameters

<i>puzzle</i>	The respective puzzle
<i>SizeOfPuzzle</i>	Puzzle size.

Here is the call graph for this function:



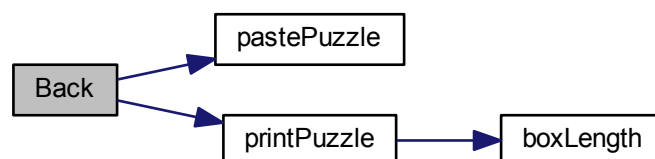
Here is the caller graph for this function:



### 2.1.1.2 void Back ( int *puzzle*, int *copy*, int *SizeOfPuzzle* )

Call back-up of last puzzle.

Here is the call graph for this function:





Here is the caller graph for this function:



### 2.1.1.3 void ChoseOptions ( )

Generate chose options interface for switch .

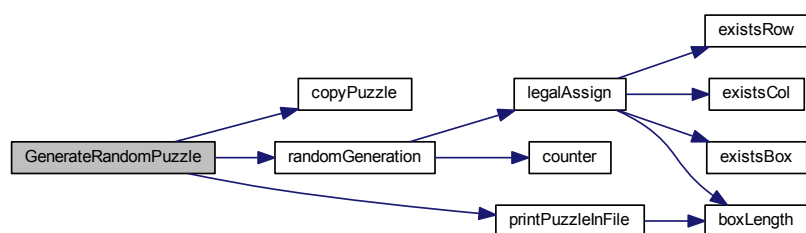
Here is the caller graph for this function:



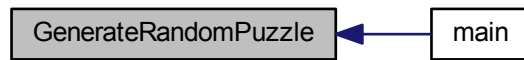
### 2.1.1.4 void GenerateRandomPuzzle ( int puzzle, int copy, FILE \* nameOfFile, int SizeOfPuzzle )

Generate puzzle.

Here is the call graph for this function:



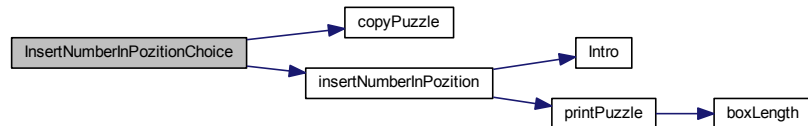
Here is the caller graph for this function:



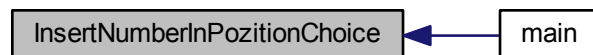
#### 2.1.1.5 void InsertNumberInPozitionChoice ( int *puzzle*, int *copy*, int *SizeOfPuzzle* )

Insert number in poztion at your choice !

Here is the call graph for this function:



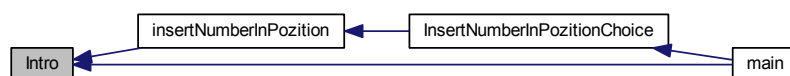
Here is the caller graph for this function:



#### 2.1.1.6 void Intro ( )

Generate intro interface.

Here is the caller graph for this function:



2.1.1.7 void SolvePuzzle ( int *puzzle*, FILE \* *fileToPrint*, int *SizeOfPuzzle* )

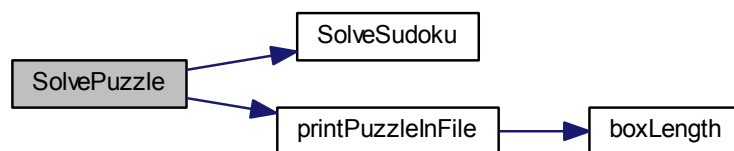
A function which will call solving parts to solve the puzzle !

The solution is printed on console display. /\*\* Generate print interface.

```
\ printPuzzle(puzzle, SizeOfPuzzle);
```

\ The solution is printed on Solutions file !

Here is the call graph for this function:



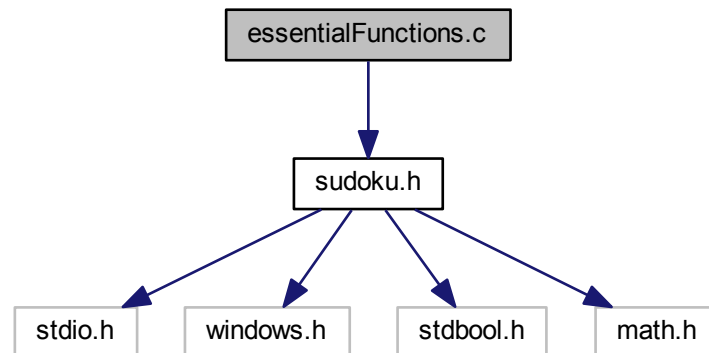
Here is the caller graph for this function:



## 2.2 essentialFunctions.c File Reference

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

Include dependency graph for essentialFunctions.c:



## Functions

- int **boxLength** (int **SizeOfPuzzle**)  
*A function which will determine box size.*
- void **copyPuzzle** (int puzzle[**SizeOfPuzzle**][**SizeOfPuzzle**], int copy[**SizeOfPuzzle**][**SizeOfPuzzle**])  
*A function to BackUp the puzzle.*
- void **pastePuzzle** (int puzzle[**SizeOfPuzzle**][**SizeOfPuzzle**], int copy[**SizeOfPuzzle**][**SizeOfPuzzle**])  
*An another function to BackUp the puzzle.*
- void **printPuzzle** (int puzzle[**SizeOfPuzzle**][**SizeOfPuzzle**], int **SizeOfPuzzle**)  
*Function which will print puzzle.*
- void **printPuzzleInFile** (FILE \*filename, int puzzle[**SizeOfPuzzle**][**SizeOfPuzzle**], int **SizeOfPuzzle**)  
*Function which will print puzzle in a file.*
- void **insertNumberInPoition** (int puzzle[**SizeOfPuzzle**][**SizeOfPuzzle**], int **SizeOfPuzzle**)  
*Function which will insert number in specificated position.*
- int **Ask** ()  
*Ask function as well as it's called just ask user which will be the size of puzzle.*
- int **counter** (int puzzle[**SizeOfPuzzle**][**SizeOfPuzzle**])  
*Counter function will return how many numbers shoud be finded in our sudoku puzzle.*
- \*param file Data file \*bool **OpenFiles** (FILE \*file)  
*A function which will check if file is open.*

### 2.2.1 Function Documentation

#### 2.2.1.1 int Ask ( )

Ask function as well as it's called just ask user which will be the size of puzzle.

Here is the caller graph for this function:



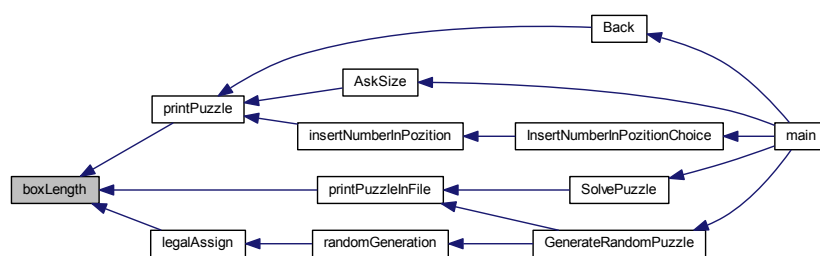
### 2.2.1.2 int boxLength ( int *SizeOfPuzzle* )

A function which will determine box size.

#### Parameters

<i>SizeOfPuzzle</i>	Puzzle Size
---------------------	-------------

Here is the caller graph for this function:



### 2.2.1.3 void copyPuzzle ( int *puzzle*[*SizeOfPuzzle*][*SizeOfPuzzle*], int *copy*[*SizeOfPuzzle*][*SizeOfPuzzle*] )

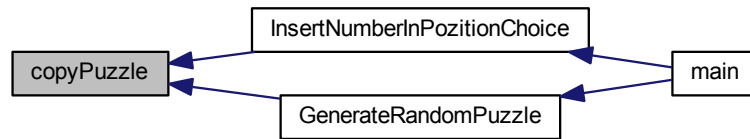
A function to BackUp the puzzle.

#### Parameters

<i>puzzle</i>	The respective puzzle
<i>copyOfPuzzle</i>	

Store the matrix into other matrix.

Here is the caller graph for this function:



#### 2.2.1.4 `int counter ( int puzzle[SizeOfPuzzle][SizeOfPuzzle] )`

Counter function will return how many numbers should be finded in our sudoku puzzle.

##### Parameters

<i>puzzle</i>	The respective puzzle
<i>SizeOfPuzzle</i>	Puzzle Size

Declare a counter.

Increase counter when poztion is not equal to 0.

Return counter value.

Here is the caller graph for this function:



#### 2.2.1.5 `void insertNumberInPozition ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int SizeOfPuzzle )`

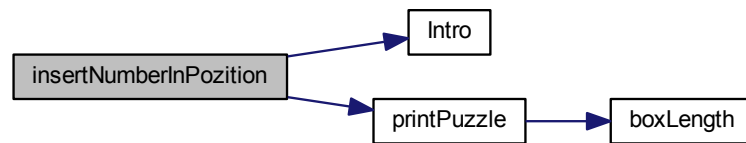
Function which will insert number in specificated position.

##### Parameters

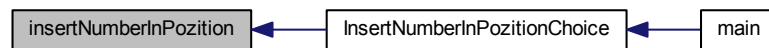
<i>puzzle</i>	The respective puzzle
<i>SizeOfPuzzle</i>	Puzzle Size

Use clear tool to delete the console

Here is the call graph for this function:



Here is the caller graph for this function:



#### 2.2.1.6 \* param file Data file\* bool OpenFiles ( FILE \* file )

A function which will check if file is open.

#### 2.2.1.7 void pastePuzzle ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int copy[SizeOfPuzzle][SizeOfPuzzle] )

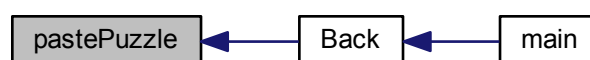
An another function to BackUp the puzzle.

##### Parameters

<i>puzzle</i>	The respective puzzle
<i>copyOfPuzzle</i>	

Assign the backup values to puzzle.

Here is the caller graph for this function:



### 2.2.1.8 void printPuzzle ( int *puzzle*[*SizeOfPuzzle*][*SizeOfPuzzle*], int *SizeOfPuzzle* )

Function which will print puzzle.

#### Parameters

<i>puzzle</i>	The respective puzzle
<i>SizeOfPuzzle</i>	Puzzle Size

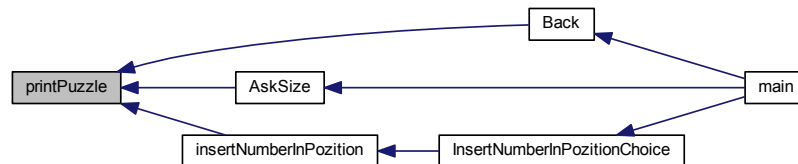
Get length of box.

Generate print interface.

Here is the call graph for this function:



Here is the caller graph for this function:



### 2.2.1.9 void printPuzzleInFile ( FILE \* *filename*, int *puzzle*[*SizeOfPuzzle*][*SizeOfPuzzle*], int *SizeOfPuzzle* )

Function which will print puzzle in a file.

#### Parameters

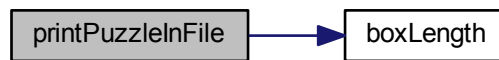
<i>filename</i>	Data file
<i>puzzle</i>	The respective puzzle
<i>SizeOfPuzzle</i>	Puzzle Size

Get length of box.

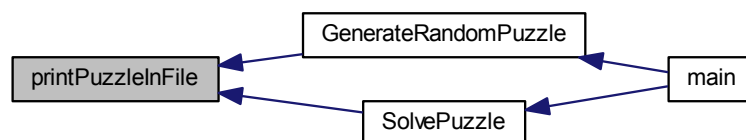
Generate print interface.



Here is the call graph for this function:



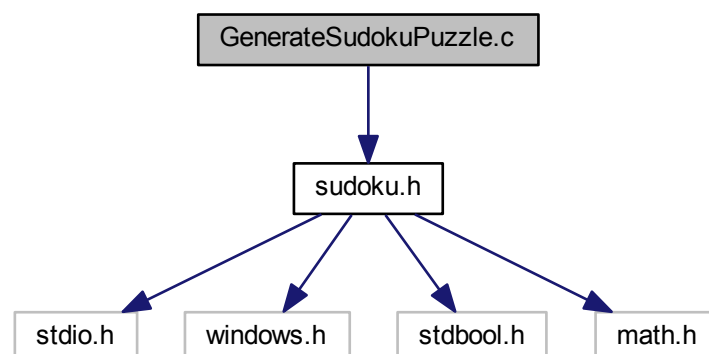
Here is the caller graph for this function:



## 2.3 GenerateSudokuPuzzle.c File Reference

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

Include dependency graph for `GenerateSudokuPuzzle.c`:



## Functions

- bool **existsRow** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int value)  
*There is Generation algorithm of our Sudoku puzzle, it is a simple generation using backtracking method !*
- bool **existsCol** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int col, int value)  
*Function which will check if the value exist on actual Column .*
- bool **existsBox** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int boxRow, int boxCol, int value)  
*Function which will check if the value exist on actual box .*
- bool **legalAssign** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int col, int value)  
*Give permission to insert value in primary matrix.*
- void **randomGeneration** (int puzzle[SizeOfPuzzle][SizeOfPuzzle])  
*There is Generation algorithm of our Sudoku puzzle, it is a simple generation using backtracking method !*

### 2.3.1 Function Documentation

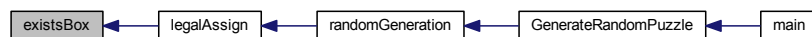
#### 2.3.1.1 bool existsBox ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int boxRow, int boxCol, int value )

Function which will check if the value exist on actual box .

##### Parameters

<i>puzzle</i>	The respective puzzle
<i>BoxRow</i>	
<i>BoxCol</i>	
<i>Value</i>	The value of number

Here is the caller graph for this function:



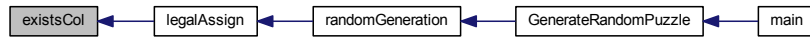
#### 2.3.1.2 bool existsCol ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int col, int value )

Function which will check if the value exist on actual Column .

##### Parameters

<i>puzzle</i>	The respective puzzle
<i>Column</i>	Respective Column
<i>Value</i>	The value of number

Here is the caller graph for this function:



#### 2.3.1.3 bool existsRow ( int *puzzle*[SizeOfPuzzle][SizeOfPuzzle], int *row*, int *value* )

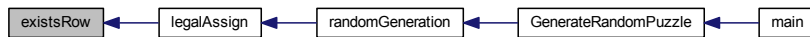
There is Generation algorithm of our Sudoku puzzle, it is a simple generation using backtracking method !

Function which will check if the value exist on actual Row .

##### Parameters

<i>puzzle</i>	The respective puzzle
<i>Row</i>	Respective Row
<i>Value</i>	The value of number

Here is the caller graph for this function:



#### 2.3.1.4 bool legalAssign ( int *puzzle*[SizeOfPuzzle][SizeOfPuzzle], int *row*, int *col*, int *value* )

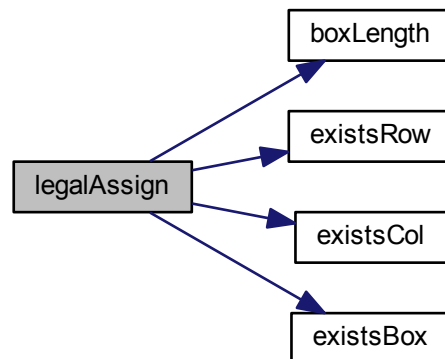
Give permission to insert value in primary matrix.

##### Parameters

<i>puzzle</i>	The respective puzzle
<i>Row</i>	Respective Row
<i>Value</i>	The value of number
<i>Value</i>	The value of number

Get size of box

Here is the call graph for this function:



Here is the caller graph for this function:



### 2.3.1.5 void randomGeneration ( int *puzzle*[SizeOfPuzzle][SizeOfPuzzle] )

There is Generation algorithm of our Sudoku puzzle, it is a simple generation using backtracking method !

#### Parameters

<i>puzzle</i>	The respective puzzle
---------------	-----------------------

Random seed by clock.

Generate random numbers on rows !

Generate random numbers on columns !

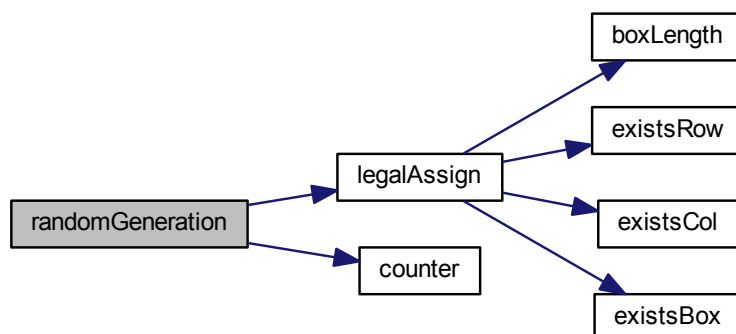
Generate a value which will fill matrix !

Get rights to assign value

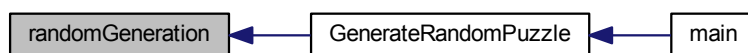
Assign the value

Declare a variable which will store how much numbers are available on matrix.

Here is the call graph for this function:



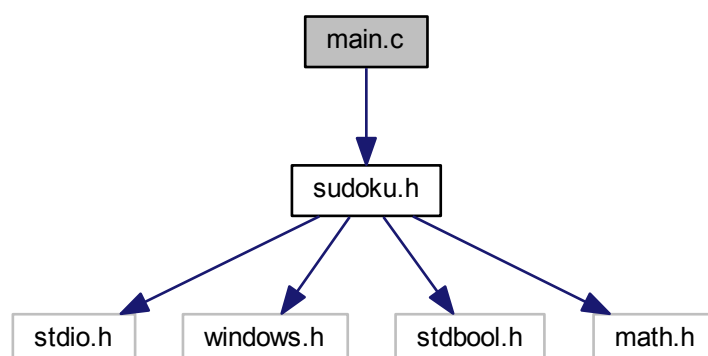
Here is the caller graph for this function:



## 2.4 main.c File Reference

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

Include dependency graph for main.c:



## Functions

- int **main** ()

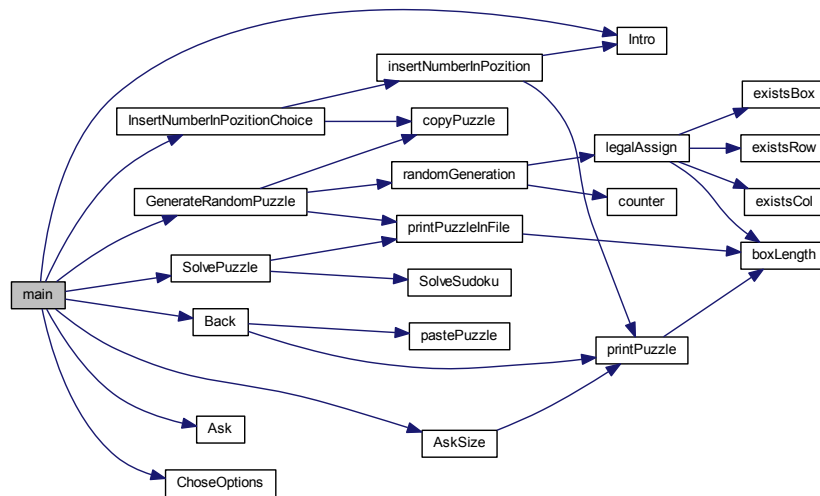
### 2.4.1 Function Documentation

#### 2.4.1.1 int main ( )

Create Interface of Program

Here is generated the puzzle full with 0

Here is the call graph for this function:



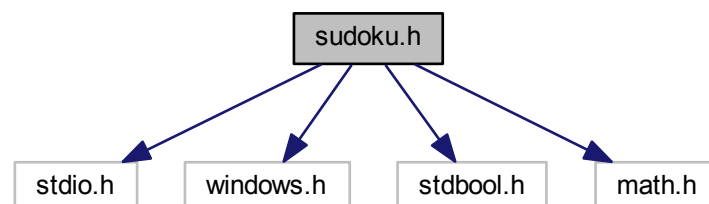
## 2.5 sudoku.h File Reference

```

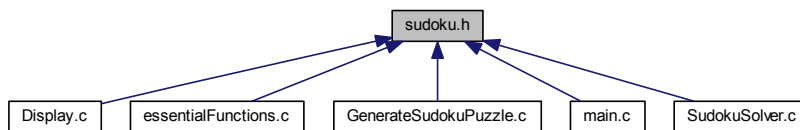
#include <stdio.h>
#include <windows.h>
#include <stdbool.h>
#include <math.h>

```

Include dependency graph for sudoku.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define UNASSIGNED 0`

## Variables

- `int SizeOfPuzzle`
- `bool GameOver`
- `bool Start`
- `int choice`
- `int numberOfSolutions`
- `int solution`
- `int iterations`

### 2.5.1 Macro Definition Documentation

#### 2.5.1.1 `#define UNASSIGNED 0`

### 2.5.2 Variable Documentation

#### 2.5.2.1 `int choice`

Interface status

#### 2.5.2.2 `bool GameOver`

Size of puzzle

#### 2.5.2.3 `int iterations`

#### 2.5.2.4 `int numberOfSolutions`

Choice status for switch statements

2.5.2.5 int SizeOfPuzzle

2.5.2.6 int solution

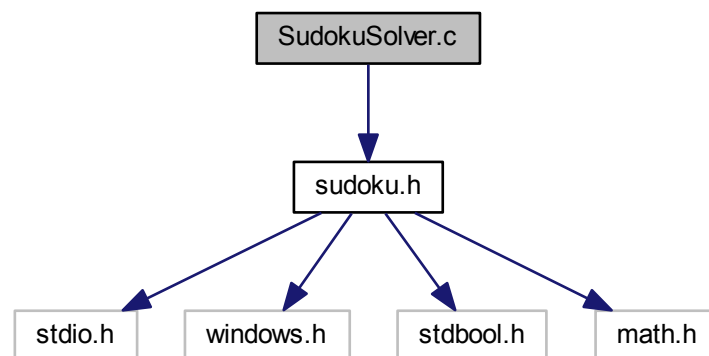
2.5.2.7 bool Start

Game status

## 2.6 SudokuSolver.c File Reference

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

Include dependency graph for SudokuSolver.c:



## Functions

- int **FindUnassigned** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int \*row, int \*col)  
*Starting counter of solution.*
- int **isSafe** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int col, int num)  
*Returns a boolean which indicates whether it will be legal to assign num to the given row,col location.*
- int **UsedInRow** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int num)  
*Returns a boolean which indicates whether any assigned entry in the specified row matches the given number.*
- int **UsedInCol** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int col, int num)  
*Returns a boolean which indicates whether any assigned entry in the specified column matches the given number.*
- int **UsedInBox** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], int BoxStartRow, int BoxStartCol, int num)  
*Returns a boolean which indicates whether any assigned entry within the specified box matches the given number.*
- int **SolveSudoku** (int puzzle[SizeOfPuzzle][SizeOfPuzzle], FILE \*fileName)  
*Takes a partially filled-in grid and attempts to assign values to all unassigned locations in such a way to meet the requirements for Sudoku solution (non-duplication across rows, columns, and boxes)*



## Variables

- **solution** = 0

### 2.6.1 Function Documentation

2.6.1.1 `int FindUnassigned ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int * row, int * col )`

Starting counter of solution.

Searches the puzzle to find an entry that is still unassigned. If found, the reference parameters row, col will be set the location that is unassigned, and true is returned. If no unassigned entries remain, false is returned.

#### Parameters

<i>puzzle</i>	The respective puzzle
<i>Row</i>	
<i>Column</i>	

2.6.1.2 `int isSafe ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int col, int num )`

Returns a boolean which indicates whether it will be legal to assign num to the given row,col location.

#### Parameters

<i>puzzle</i>	The respective puzzle
<i>Row</i>	
<i>Column</i>	
<i>Number</i>	

Check if 'num' is not already placed in current row, current column and current box

2.6.1.3 `int SolveSudoku ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], FILE * fileName )`

Takes a partially filled-in grid and attempts to assign values to all unassigned locations in such a way to meet the requirements for Sudoku solution (non-duplication across rows, columns, and boxes)

#### Parameters

<i>puzzle</i>	The respective puzzle
<i>file</i>	Data file

If there is no unassigned location, we are done

Starting counter of iterations

Consider digits 1 to SizeOfPuzzle

If looks promising

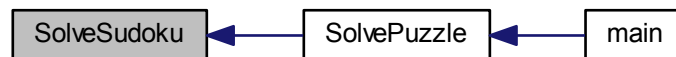
Make tentative assignment

Return, if success

Failure, unmake & try again

This triggers backtracking

Here is the caller graph for this function:



#### 2.6.1.4 `int UsedInBox ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int BoxStartRow, int BoxStartCol, int num )`

Returns a boolean which indicates whether any assigned entry within the specified box matches the given number.

##### Parameters

<i>puzzle</i>	The respective puzzle
<i>BoxStartRow</i>	
<i>BoxStartCol</i>	
<i>Number</i>	

#### 2.6.1.5 `int UsedInCol ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int col, int num )`

Returns a boolean which indicates whether any assigned entry in the specified column matches the given number.

##### Parameters

<i>puzzle</i>	The respective puzzle
<i>Column</i>	
<i>Number</i>	

#### 2.6.1.6 `int UsedInRow ( int puzzle[SizeOfPuzzle][SizeOfPuzzle], int row, int num )`

Returns a boolean which indicates whether any assigned entry in the specified row matches the given number.

## Parameters

<i>puzzle</i>	The respective puzzle
<i>Row</i>	
<i>Number</i>	

## 2.6.2 Variable Documentation

### 2.6.2.1 solution = 0



# Index

- Ask
  - essentialFunctions.c, 8
- AskSize
  - Display.c, 4
- Back
  - Display.c, 4
- boxLength
  - essentialFunctions.c, 9
- choice
  - sudoku.h, 19
- ChoseOptions
  - Display.c, 5
- copyPuzzle
  - essentialFunctions.c, 9
- counter
  - essentialFunctions.c, 10
- Display.c, 3
  - AskSize, 4
  - Back, 4
  - ChoseOptions, 5
  - GenerateRandomPuzzle, 5
  - InsertNumberInPozitionChoice, 6
  - Intro, 6
  - SolvePuzzle, 6
- essentialFunctions.c, 7
  - Ask, 8
  - boxLength, 9
  - copyPuzzle, 9
  - counter, 10
  - insertNumberInPozition, 10
  - OpenFiles, 11
  - pastePuzzle, 11
  - printPuzzle, 11
  - printPuzzleInFile, 12
- existsBox
  - GenerateSudokuPuzzle.c, 14
- existsCol
  - GenerateSudokuPuzzle.c, 14
- existsRow
  - GenerateSudokuPuzzle.c, 15
- FindUnassigned
  - SudokuSolver.c, 21
- GameOver
  - sudoku.h, 19
- GenerateRandomPuzzle
  - Display.c, 5
  - GenerateSudokuPuzzle.c, 13
  - existsBox, 14
  - existsCol, 14
  - existsRow, 15
  - legalAssign, 15
  - randomGeneration, 16
- insertNumberInPozition
  - essentialFunctions.c, 10
- InsertNumberInPozitionChoice
  - Display.c, 6
- Intro
  - Display.c, 6
- isSafe
  - SudokuSolver.c, 21
- iterations
  - sudoku.h, 19
- legalAssign
  - GenerateSudokuPuzzle.c, 15
- main
  - main.c, 18
- main.c, 17
  - main, 18
- numberOfSolutions
  - sudoku.h, 19
- OpenFiles
  - essentialFunctions.c, 11
- pastePuzzle
  - essentialFunctions.c, 11
- printPuzzle
  - essentialFunctions.c, 11
- printPuzzleInFile
  - essentialFunctions.c, 12
- randomGeneration
  - GenerateSudokuPuzzle.c, 16
- SizeOfPuzzle
  - sudoku.h, 19
- solution
  - sudoku.h, 20
  - SudokuSolver.c, 23
- SolvePuzzle
  - Display.c, 6
- SolveSudoku

- SudokuSolver.c, 21
- Start
  - sudoku.h, 20
- sudoku.h, 18
  - choice, 19
  - GameOver, 19
  - iterations, 19
  - numberOfSolutions, 19
  - SizeOfPuzzle, 19
  - solution, 20
  - Start, 20
  - UNASSIGNED, 19
- SudokuSolver.c, 20
  - FindUnassigned, 21
  - isSafe, 21
  - solution, 23
  - SolveSudoku, 21
  - UsedInBox, 22
  - UsedInCol, 22
  - UsedInRow, 22
- UNASSIGNED
  - sudoku.h, 19
- UsedInBox
  - SudokuSolver.c, 22
- UsedInCol
  - SudokuSolver.c, 22
- UsedInRow
  - SudokuSolver.c, 22