

## Introduction

In traditional Sudoku, we have a game board that is 9 cells wide by 9 cells tall. However, for this tiny-Sudoku, we are breaking it down into only a 3x3 space.

On the board, each cell can have a number between 1 and 9, but each number can only show up once. In our program, we will check to make sure we're not placing any duplicate numbers on the game board.

	0	1	2
0	2   1   6		
1	3   7   4		
2	5   8   9		

## Code Files

The starter files comes with two separate projects – the **tester** and the **program**. First you will be working with the tester project to make sure your functions run as intended. Once your functions work right, you can use it in the program and test the game as a whole.

You should have the following files in your Tiny Sudoku project folder.

- `sudoku_functions.hpp` This is the file you work with
- `sudoku_program.cpp` The program code, already written
- `Tester.hpp` The test functions, already written
- `tester_program.cpp` The test program, already written
- `utilities.hpp` Additional C++ utilities, already written
- `cuTEST` folder This is a testing framework I've written, you won't modify any of these files.
  - `Menu.hpp`
  - `StringUtil.hpp`
  - `TesterBase.cpp`
  - `TesterBase.hpp`

**Make sure that you've downloaded the starter files off Desire2Learn!**

Additionally, there is a **demos** folder that contains executable files of the program that you can run to test out its functionality.

## Table of Contents

Introduction.....	1
Code Files.....	1
Working with Arrays.....	3
Working with the Tester.....	4
sudoku_functions.hpp.....	7
Functions with tests.....	7
PlaceNumberInCell.....	7
CellIsTaken.....	8
ResetGrid.....	8
NumberAlreadyUsed.....	9
AllCellsTaken.....	10
Functions without tests.....	11
GetUserInput.....	11
GetRow.....	11
GetColumn.....	11
GetNumber.....	12
Working with the Program.....	12
Extra Credit.....	13
Example Screenshots.....	14
Grading Breakdown.....	15

## Working with Arrays

The array we're working with, **grid**, is a 2D array. `int grid[3][3];`

It is the representation of our 3x3 grid for a single Sudoku block. Because the grid is 3x3, this means that the valid indices range from 0 to 2.

	Column 0	Column 1	Column 2
Row 0	<code>grid[0][0]</code>	<code>grid[0][1]</code>	<code>grid[0][2]</code>
Row 1	<code>grid[1][0]</code>	<code>grid[1][1]</code>	<code>grid[1][2]</code>
Row 2	<code>grid[2][0]</code>	<code>grid[2][1]</code>	<code>grid[2][2]</code>

For this project, you will be using **for loops** a lot to iterate through all 9 cells of the grid. A single for loop won't be sufficient; you could only traverse one row, or one column. In order to go through all the rows and columns of the 2D array, we need two for loops. Therefore, you will use the following nested loops a lot:

```
for ( int r = 0; r < 3; r++ )
{
    for ( int c = 0; c < 3; c++ )
    {
        // Do something with grid[r][c]
    }
}
```

For example, if you wanted to set all items in the grid to 0, you would do so like this:

```
for ( int r = 0; r < 3; r++ )
{
    for ( int c = 0; c < 3; c++ )
    {
        grid[r][c] = 0;
    }
}
```

Additionally, if you have values for the row and column stored in some variables, you could directly access a grid item like:

```
grid[row][column] = 0;
```

## Working with the Tester

There is a separate **Tester** program that you should be working in while writing the functions (in `sudoku_functions.hpp`). When you run the tester, it will generate output text files with the results – it will run multiple tests, and tell you which ones passed and which ones failed.

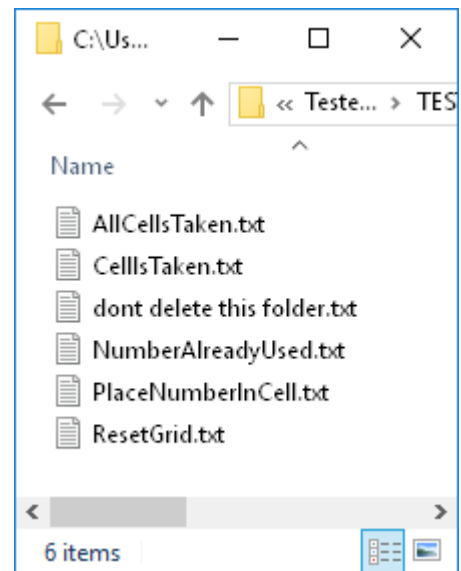
Sometimes, some tests will pass, even if your function logic is wrong. This is why there are multiple tests. Once all tests pass for a given function, that function is probably working correctly.


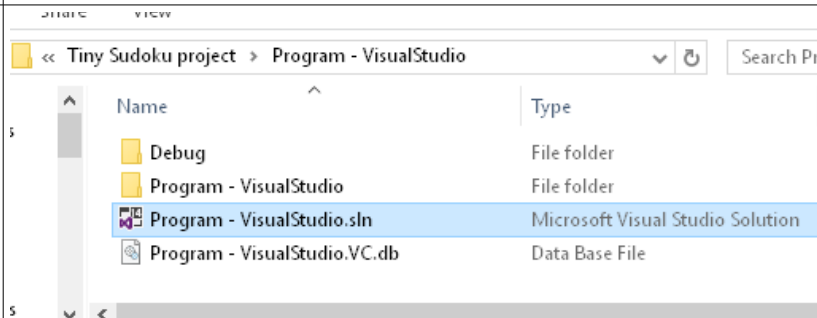
Once you run the tester, it will generate a set of text files with the function results, within a folder called **TEST\_RESULTS**. This folder will be generated somewhere in the tester project folder.

- Code::Blocks:  
Tiny Sudoku project/Tester – CodeBlocks/TEST\_RESULTS
- Visual Studio:  
Tiny Sudoku project/Tester – Visual Studio/Tester – Visual Studio/TEST\_RESULTS

Let's step through implementing **one function** to show you how this works.

Open up the project for the Tester program.



Code::Blocks	Visual Studio
 <p>Tester - CodeBlocks.cbp</p> <p>Tester – CodeBlocks.cbp</p>	

Build and run the program. The output window won't display any text, but afterwards, the text files will be generated in the **TEST\_RESULTS** folder.

Open up the file for **CellsTaken.txt**

By default, the function it is testing is set to only return **true**, so some tests may pass but most will fail. The output file will look like this.

The header tells you which function is being tested.

```
*****
TEST SET: CellIsTaken
PREREQUISITE FUNCTIONS:
*****
```

Then, several tests are run, and each has a header that describes what it tests.

```
-----
Test 1:    Check to see if CellIsTaken() returns
false for all cells in an empty grid
-----
```

If a test passes, it will have the label:

```
(Test 1 PASSED)
```

```
*****
TEST SET: CellIsTaken
PREREQUISITE FUNCTIONS:
*****
```

```
-----
Test 1:    Check to see if CellIsTaken()
returns false for all cells in an empty grid
-----
```

```
(Test 1 PASSED)
```

```
-----
Test 2:    Check to see if CellIsTaken()
returns true for a full grid
-----
```

```
(Test 2 FAILED)  CellIsTaken_Test2
With full grid, checked row 0 column 0,
CellIsTaken() should return TRUE but returned
FALSE.
```

```
(Test 2 FAILED)  CellIsTaken_Test2
With full grid, checked row 0 column 1,
CellIsTaken() should return TRUE but returned
FALSE.
```

But if it fails, it will display a failure message, and a description of what kind of test was run, what it **expected to happen** and what **actually happened (which caused the test to fail)**:

```
(Test 2 FAILED)  CellIsTaken_Test2
With full grid, checked row 0 column 0, CellIsTaken() should return TRUE but returned FALSE.
```

Pay attention to the failure messages! It will let you know what went wrong, to help you figure out how to correctly write the functions!

Let's fix that function so all the tests for it pass.

Now, let's look in `sudoku_functions.hpp` at the `CellIsTaken` function. By default, it looks like this:

```
bool CellIsTaken( int grid[3][3], int row, int column )
{
    return false;
}
```

We need to add some logic so it actually works correctly. This function should return **true** if a certain (row, column) of the grid is something other than 0. If the cell at (row, column) is actually 0, then it will return false.

So, add in the following code:

```
bool CellIsTaken( int grid[3][3], int row, int column )
{
    if ( grid[row][column] == 0 )
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

And then re-run the tester. The output should now say that all 3 tests passed.

```
*****
TEST SET: CellIsTaken
PREREQUISITE FUNCTIONS:
*****

-----
Test 1:    Check to see if CellIsTaken() returns false for all cells in an empty grid
(Test 1 PASSED)

-----
Test 2:    Check to see if CellIsTaken() returns true for a full grid
(Test 2 PASSED)

-----
Test 3:    Check to see if CellIsTaken() returns true or false correctly on partially-filled grid
(Test 3 PASSED)

(PASS) 3 passed out of 3 tests
```

As you work on each function, you can run the tests to make sure it is working correctly, before you get into the main program.

## sudoku\_functions.hpp

You will be doing all your coding in this one file.

**WORK ON ONE FUNCTION AT A TIME**, then use the tester to check your results. If your tests fail, go back and modify the function until you get it working properly.

After all your functions work, you will use it in the actual program itself.

**Note: Not all functions have tests.** The functions that use **cin** don't have tests associated with them!

---

### Functions with tests

#### PlaceNumberInCell

Parameter name	Data type	Description
grid	2D integer array, 3x3	The grid structure
row	integer	The row to place the number (between 0 and 2)
column	integer	The column to place the number (between 0 and 2)
number	integer	The number value (between 1 and 9)

Given the **row** and **column** passed in, place the **number** given into that cell of the **grid**.

Recall from the [Working with Arrays](#) section, the different cells of the grid:

	Column 0	Column 1	Column 2
Row 0	grid[0][0]	grid[0][1]	grid[0][2]
Row 1	grid[1][0]	grid[1][1]	grid[1][2]
Row 2	grid[2][0]	grid[2][1]	grid[2][2]

You will use the **row** and **column** variables to access a specific cell, and set that cell to the **number** passed in.

Afterward, you should also display a message to the user, like:

```
cout << number << " placed at " << row << ", " << column << endl;
```

## CellIsTaken

Parameter name	Data type	Description
grid	2D integer array, 3x3	The grid structure
row	integer	The row (between 0 and 2)
column	integer	The column (between 0 and 2)

For this function, it will look at the grid item at position (*row*, *column*).

- If this cell has the # 0 in it, it is considered empty, and the function will return **false**.
- If this cell has some number other than 0 in it, it is considered full (or taken), and the function will return **true**.

**Big Picture:** This function is used so that, when the player wants to place a number in cell (*row*, *column*), we can first check to see if it is empty before we place that number in – so that we don't overwrite any existing values.  
(This logic is handled in the *main()* program, so you won't implement that here.)

---

## ResetGrid

Parameter name	Data type	Description
grid	2D integer array, 3x3	The grid structure

This function is responsible for *resetting* our game board grid. When it is run, it should set all cells to 0.

You will use a nested for loop, as mentioned in the [Working with Arrays](#) section, to investigate every cell of the grid.

```
for ( int r = 0; r < 3; r++ )
{
    for ( int c = 0; c < 3; c++ )
    {
        // Do something with grid[r][c]
    }
}
```

**Big Picture:** In the *main()* program, once a round ends (the board has been filled), it will ask the user if they want to play again. If they do want to play again, we need to *Reset()* the board so we are ready for another round.  
(This logic is handled in the *main()* program, so you won't implement that here.)



## NumberAlreadyUsed

Parameter name	Data type	Description
grid	2D integer array, 3x3	The grid structure
number	Integer	A number (between 1 and 9)

This function will check **all cells of the grid**, searching for the **number**.

- If the **number** is found, it will return **true**.
- Otherwise, if the **number** is not found in the grid, it will return **false**.

You will use a nested for loop, as mentioned in the Working with Arrays section, to investigate every cell of the grid.

```
for ( int r = 0; r < 3; r++ )
{
    for ( int c = 0; c < 3; c++ )
    {
        // Do something with grid[r][c]
    }
}

// All cells have been checked
```

**Big Picture:** In the grid, each number (1 – 9) will only be allowed once. We don't want duplicates in the grid, so this function is used to make sure we don't add duplicates.

*(This logic is handled in the main() program, so you won't implement that here.)*

**Challenge:** You might be able to figure out the logic to return **true** if the number is found, but the logic on **when** to return **false** is usually more difficult for new programmers to figure out.

Hint 1: Within your nested for loops, if you find the number, you will return true **inside the nested for loops**.

Hint 2: You will **not return false inside any of the for loops!** Otherwise your function logic will be incorrect.

## AllCellsTaken

Parameter name	Data type	Description
grid	2D integer array, 3x3	The grid structure

This function goes through all cells of the grid, checking to see if they're all taken. (If a cell is 0, then it is "empty" or not taken.)

- If there EXISTS at least one cell that is 0, then return **false**. (Do this within the for loop once the empty cell is discovered.)
  - You can call the **CellIsTaken** function, which you should have already implemented, to see if that specific cell is not taken. (**CellIsTaken( grid, r, c ) == false**)
- If ALL cells are not 0, then return **true**. (You will only do this outside of all for loops, once we are certain that all cells have been checked.)

You will use a nested for loop, as mentioned in the Working with Arrays section, to investigate every cell of the grid.

```
for ( int r = 0; r < 3; r++ )
{
    for ( int c = 0; c < 3; c++ )
    {
        // Do something with grid[r][c]
    }
}

// All cells have been checked
```

**Big Picture:** Once all cells are full, the main() game will know that the round is over, and end the game.  
(This logic is handled in the main() program, so you won't implement that here.)

## Functions without tests

These functions do not have tests associated with them, because of the **cin** statements.

### GetUserInput

Parameter name	Data type	Description
min	integer	The minimum possible value for the user input
max	integer	The maximum possible value for the user input

1. Create an integer variable called **choice**
  2. Prompt the user to enter something, such as with a “>> ” prompt.
  3. Get the user’s input via **cin** and store it in **choice**.
  4. While the user’s choice is invalid... (choice is less than min or greater than max)
    1. Display a message “Invalid choice, try again”
    2. Get the user’s input via **cin** and store it in **choice**.
  5. Outside of the loop, **return choice**.
- 

### GetRow

No parameters, returns **int**

1. Display a message to the user, “Enter row:”
2. Create an integer variable called **row**.
3. Call the **GetUserInput** function, with 0 as min and 2 as max. Store the result in the **row** variable.
4. Return **row**.

```
int GetRow()  
{  
    cout << "Enter row (0 - 2):   ";  
    int row = GetUserInput( 0, 2 );  
    return row;  
}
```

### GetColumn

No parameters, returns **int**

1. Display a message to the user, “Enter column:”
2. Create an integer variable called **column**.
3. Call the **GetUserInput** function, with 0 as min and 2 as max. Store the result in the **column** variable.
4. Return **column**.

---


## GetNumber

No parameters, returns **int**

1. Display a message to the user, “Enter number:”
  2. Create an integer variable called **number**.
  3. Call the **GetUserInput** function, with 1 as min and 9 as max. Store the result in the **number** variable.
  4. **Return number.**
- 

## Working with the Program

After you’ve tested the functions with the tester, you will open up the program project.

Code::Blocks	Visual Studio										
 Program - CodeBlocks. cbp  Program – CodeBlocks.cbp	<table><thead><tr><th>name</th><th>type</th></tr></thead><tbody><tr><td>Debug</td><td>File folder</td></tr><tr><td>Program - VisualStudio</td><td>File folder</td></tr><tr><td>Program - VisualStudio.sln</td><td>Microsoft Visual Studio Solution</td></tr><tr><td>Program - VisualStudio.VC.db</td><td>Data Base File</td></tr></tbody></table> Program – VisualStudio.sln	name	type	Debug	File folder	Program - VisualStudio	File folder	Program - VisualStudio.sln	Microsoft Visual Studio Solution	Program - VisualStudio.VC.db	Data Base File
name	type										
Debug	File folder										
Program - VisualStudio	File folder										
Program - VisualStudio.sln	Microsoft Visual Studio Solution										
Program - VisualStudio.VC.db	Data Base File										

When you build and run the program, the Tiny Sudoku program should start and you should be able to play it.

Look at the [Example Screenshots](#) for more information.

On D2L, you may also be able to download an **executable demo** of the complete program (without source code) to test.

```
*****
* Mini-Sudoku *
*****

      0   1   2
0      | 1 |   | 3 |
      -----
1      | 6 | 7 | 4 |
      -----
2      |   | 8 |   |

Enter row (0 - 2):  >> █
```

## Extra Credit

If you want to add additional features, you can for some extra credit.

---

## Grading Breakdown

Breakdown			
Score			
Item	Score (0-5)	Task weight	Weighted score
PlaceNumberInCell	5	10.00%	10.00%
CellIsTaken	5	10.00%	10.00%
ResetGrid	5	15.00%	15.00%
NumberAlreadyUsed	5	20.00%	20.00%
AllCellsTaken	5	20.00%	20.00%
GetUserInput	5	10.00%	10.00%
GetRow	5	5.00%	5.00%
GetColumn	5	5.00%	5.00%
GetNumber	5	5.00%	5.00%

Score totals	100.00%	100.00%
--------------	---------	---------

Penalties			
Item	Score (0-4)	Max penalty	Weighted penalty
Syntax errors (doesn't build)	0	-50.00%	0.00%
Logic errors	0	-10.00%	0.00%
Run-time errors	0	-10.00%	0.00%
Memory errors (leaks, bad memory access)	0	-10.00%	0.00%
Ugly code (bad indentation, no whitespacing)	0	-5.00%	0.00%
Ugly UI (no whitespacing, no prompts, hard to use)	0	-5.00%	0.00%
Not citing code from other sources	0	-100.00%	0.00%

Penalty totals	0.00%
----------------	-------

Totals	Final score:	100.00%
--------	--------------	---------

## Example Screenshots

**Entering a number that is already stored in one of the cells of the grid.**

```
Program - CodeBlocks
File Edit View Search Terminal Help
*****
*****
* Mini-Sudoku *
*****

      0  1  2
0      |  | 4 | 2 |
-----
1      | 7 |  |  |
-----
2      | 6 | 1 | 5 |

Enter row (0 - 2):  >> 0
Enter column (0 - 2): >> 0
Enter number (1 - 9): >> 1

1 is already in the grid!
Enter number (1 - 9): >> █
```

**Entering a row, column that is already taken up by a number.**

```
Program - CodeBlocks
File Edit View Search Terminal Help
*****
*****
* Mini-Sudoku *
*****

      0  1  2
0      | 4 | 7 |  |
-----
1      | 6 | 2 | 3 |
-----
2      | 5 |  |  |

Enter row (0 - 2):  >> 0
Enter column (0 - 2): >> 0

Cell is already taken
Enter row (0 - 2):  >> █
```

**Displaying a message that number was placed.**

```
Program - CodeBlocks
File Edit View Search Terminal Help
*****
*****
* Mini-Sudoku *
*****

      0  1  2
0      | 2 | 1 | 6 |
-----
1      | 3 | 7 | 4 |
-----
2      |  |  |  |

Enter row (0 - 2):  >> 2
Enter column (0 - 2): >> 0
Enter number (1 - 9): >> 5
5 placed at 2, 0

Press ENTER to continue...
█
```

**Game finished, asking to play again.**

```
Program - CodeBlocks
File Edit View Search Terminal Help

      0  1  2
0      | 2 | 1 | 6 |
-----
1      | 3 | 7 | 4 |
-----
2      | 5 | 8 | 9 |

All cells filled! Play again? (y/n): █
```