Group, Open Notes Exercise on Arrays

1. Form yourselves into groups, with each group having at least 3 members and at most 4 members. Should there be any group with more or less members than is stated, your teacher may split or merge, as necessary.

2. You can discuss only with your group mates on how to approach the solution. If you will check the internet of what Sudoku is, you are only to look at sample sudoku puzzles, sample solved sudoku puzzles, and the definition. No searches (anywhere) should be done on algorithm on how to solve or how to verify correctness of sudoku solution should be done.

3. During the session, the teacher will visit each breakout room to answer any questions (including any clarifications on topics previously discussed), if there are any. Discussions by the group will also be observed (but not recorded). It is recommended that a member share the screen. However, an alternative is to post the repl.it link that the group is collaborating on (also sent privately to the teacher in the Chat of the breakout room). Initial feedback will only be given during the class session.

4. Code submitted should follow coding standards and follow requirements as stated below. Internal documentation is desirable, but not required. You may add additional functions as you deem necessary, but these should not replace or modify any of the given constraints/requirements. You are not allowed to modify the file sudoku.h, where the function prototypes of the functions you need to implement are defined in.

5. You are to create your solutions from the given template file. This is the program you will compile and run. Ideally, you use input redirection to test your program. Testing ideally is done in parts too, so you can isolate the issue before moving on to the next requirements. A sample set of inputs is provided in sampleSudoku.txt.

The program should:

1.) Get inputs from the user to fully fill up a 9 x 9 matrix. Access to the array should be row-wise (row-major). This set of inputs represents a user's answer to a Sudoku problem. The following functions should be created in relation to this requirement:

```
#define SIZE 9

/* This function returns 1 if the parameter is within the range of 1 to 9,
inclusive.  Otherwise, the function returns 0. */
int isInRange(int num);

/* This function gets input for the 9 x 9 2D array.  The array then
represents the solution of the user to a Sudoku puzzle. Part of the solution
for this function is call/s to isInRange() function to check that each input
is valid. The input is taken in row-wise.

Do NOT put any printf() statement here. */
void getInput(int aMatrix[][SIZE])
{  int row, col;
 /* you are NOT allowed to declare additional variables. */
}
```

2.) Display the contents of the matrix. Display should be row-wise (assisted via function call to displayRow() which you will also create). The resulting display should look like a table. There is no need to display the lines, but there should be spaces to separate groups of 3 rows and 3 columns. Sample result should be something like the following:

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

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

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

The following functions should be created in relation to this requirement:

```
/* This function displays the values in the array, which represents 1 row of
values. */
void displayRow(int aRow[]);

/* This function displays the values of the 2D array, row-wise.  Showing the
entries as a table. Part of the solution for this function is call/s to
function displayRow(). */
void displayAll(int aMatrix[][SIZE]);
```

3.) Determine if the given matrix is a correctly solved Sudoku puzzle or not. A Sudoku puzzle is correctly solved if all of the following conditions are met:

- Each row contains all the values 1 to 9.
- Each column contains all the values 1 to 9.
- Each 3 x 3 box contains all the values 1 to 9.

The following functions should be created in relation to this function. You are also tasked to complete the main() function. **Hint:** Once you figure out an algorithm to determine that all values 1 to 9 is in the collection, you can apply the same algorithm to all three check functions; the only difference may just be in the accessing.

```
/* This function returns 1 when the 1-D array aData, contains all the values
1 to 9.  Otherwise, this function returns 0.  */
int checkrow(int aData[]);

/* This function returns 1 when the column nColInd of the 2D array aMatrix
contains all the values 1 to 9. Otherwise, this function returns 0. That is,
this function only returns the result of checking 1 column, where the column
being checked is the one indicated by nColInd (representing column index) */
int checkcol(int aMatrix[][SIZE], int nColInd);

/* This function returns 1 when the 3 x 3 box starting at row index nRow and
the column index nCol of the 2D array aMatrix contains all the values 1 to
9.  Otherwise, this function returns 0.  That is, this function only returns
the result of checking one of the 3 x 3 box. For example, if nRow is 3 and
nCol is 6, this function checks the if 1 to 9 are in [3,6], [3,7], [3,8],
[4,6], [4,7], [4,8], [5,6], [5,7], [5,8]. */
int checkbox(int aMatrix[][SIZE], int nRow, int nCol);

int main()
{
     /* refer to template file for the details */

}
```