Group, Open Notes Exercise on Arrays

1. Form yourselves into groups, with each group having at least 3 members and at most 5 members. Should there be any group with more or less members than is stated, split or merge, as necessary. You can discuss only with your group mates.

2. By the end of the session, each group is to submit the **printout** of their source code. Upload softcopy as attachment via Submit Assignment in Canvas (i.e., 1 member uploads softcopy and indicate names of all members as comment. File should also contain names of members as part of comments.

3. 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.

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. */
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()
{   int aMatrix[SIZE][SIZE];

    /* write statements to call getInput(), displayAll().
       --For testing purposes, when you have already verified that the
    getInput() and displayAll() are working properly, comment out these
    function calls and modify the declaration for the matrix above to
    include initialization (so that you do not need to always input a total
    of 81 numbers just to proceed to checking the next functions. */

    /* call functions checkrow(), checkcol(), checkbox() as part of your
    solution to determine if the matrix is a correctly solved Sudoku puzzle.
    Display a message Sudoku! when the matrix is a proper solution for
    Sudoku.  Otherwise, display Wrong Solution. */

    return 0;
}
```