2 Dimensional Arrays and Arrays of Arrays

1. Complete the documentation for the following function. Note that the function name and some of the identifiers in this function have been poorly chosen intentionally – rename them. Assume that `NUMCOLS` has been defined to be the number of columns in the array passed as an argument.

```
/*
 * Purpose: finds and returns the biggest number in row_num of data
 *
 * Parameters:  int data[][NUMCOLS], a 2D array,
 *              int row_num - a valid row index, >=0 and <number of rows in data
 *
 * Returns: int - the biggest value in row_num
 */
int get_biggest_in_row( int data[][ NUMCOLS ], int row_num ) {
   int index;
   int biggest = data[ row_num ][ 0 ];

   for( index = 1; index < NUMCOLS; index++ ) {
      if( data[ row_num ][ index ] > ret_val )
         biggest = data[ row_num ][ index ];
   }

   return biggest;
}
```

2. Complete the function `sum_column` according to the given documentation. Assume that `NUMCOLS` has been defined to be the number of columns in `data`.

```
/* purpose: calculates and returns the sum of values in col_num of data
 * parameters: int data[][NUM_COLS],
 *  int num_rows, number of rows in data, >=0
 *  int col_num, column number of values in data to be summed, >=0 and <NUM_COLS
 * returns: int, the sum
 */
int sum_column(int data[][ NUMCOLS ], int num_rows, int col_num) {
    int index;
    int sum = 0;

    for( index = 0; index < num_rows; index++ ) {
        sum += data[ index ][ col_num ];
    }
    return sum;
}
```

3. Re-write the function from the previous question as `sum_column_ptr` so that is takes `data` as an array of pointers (an array of arrays) instead of a 2 dimensional array. You can assume that each row in data has the same number of columns.

```
/* purpose: calculates and returns the sum of values in col_num of data
 * parameters: int* data[], an array of arrays of equal length
 *  int num_rows, number of rows in data, >=0
 *  int col_num, column number of values in data to be summed,
 *              >=0 and < the number of columns in data
 * returns: int, the sum
 */
int sum_column_ptr(int* data[], int num_rows, int col_num ) {
    int index;
    int sum = 0;

    for( index = 0; index < num_rows; index++ ) {
         sum += data[ index ][ col_num ];
    }

    return sum;
}
```

4. Complete the `main` function below to test the `sum_column` and the `sum_column_ptr` functions.

```
int main( void ) {
    int data[][3] = { { 1, 4, -5 }, { 5, -2, 6 }, { -2, 7, 5 } };
    int a1[3] = {1, 2, 3};
    int a2[3] = {4, 5, 6};
    int a3[3] = {7, 8, 9};

    int* data_ptrs[3] = { a1, a2, a3 };
    int sum = sum_column(data, 3, 2);
    int sum_ptrs = sum_column_ptr(data_ptrs, 3, 2);

    printf( "sum should be 6: %d, sum_ptrs should be 18: %d\n", sum, sum_ptrs );

    return 0;
}
```

5. What value is returned when the following function is called? Note that some of the identifiers in this function have been poorly chosen intentionally – rename them. Assume that all rows of `data` have the same number of columns and that `count_rows` is a value between 1 and the number of rows in the array data.

```
/* purpose: calculates and returns the sum of values the first count_rows of data
 * parameters: int data[][NUM_COLS],
 *  int count_rows,  >=0 and < the number of rows in data
 *  int num_cols, the number of columns in data, >=0 and <NUM_COLS
 * returns: int, the sum
 */
int sum_rows( int data[][NUMCOLS], int count_rows, int num_cols ) {
   int row, col;
   int sum = 0;

   for( row = 0; row < count_rows; row++ ) {
      for( col = 0; col < num_cols; col++ ) {
         sum += data[ row ][ col ];
      }
   }

   return sum;
}
```

6. When a call to the following function ends, what value is stored in `anonymous[ i ], 0 <= i < num_rows`, where `num_rows` is the number of rows in the array `data`? In other words, how would you describe the values stored in `anonymous[0],anonymous[1],anonymous[2]`

```
/* purpose: calculates and stores the sum of each row of data
 *            to the corresponding index of sums_of_rows
 * parameters: int data[][NUM_COLS],
 *  int num_rows, the number of rows in data, >=0
 *  int num_cols, the number of columns in data, >=0 and <NUM_COLS
 *  int sums_of_rows, result array, capacity is >= num_rows
 * returns: nothing
 */
void sum_rows(int data[][NUM_COLS], int num_rows, int num_cols, int sums_of_rows[] ) {
   int row, col, sum;

   for( row = 0; row < num_rows; row++ ) {
       sum = 0;
       for( col = 0; col < num_cols; col++ ) {
           sum += data[ row ][ col ];
       }
       sums_of_rows[ row ] = sum;
   }
}
```

7. Design a function that takes an array of type `int*`, a number of columns, a row index and an integer threshold value as parameters. Each index of the array is an array with length equal to the given number of columns. The function should return the number of values in the specified row of the array that are smaller than the given threshold value.

```
/* purpose: counts the number of values in the given row of data
 *          that are below the given threshold
 * parameters: int* data[], an array of arrays of equal length
 *  int num_cols, number of columns in data, >=0
 *  int row, row number of values in data to be counted,
 *           >=0 and < the number of rows in data
 * returns: int, the count
 */
int count_below_in_row(int* data[], int num_cols, int row, int threshold) {
    int col;
    int count = 0;
    for( col = 0; col < num_cols; col++ ) {
        if (data[row][col] < threshold) {
            count++;
        }
    }
    return count;
}
```

8. Write a function `sum_matrices` that takes five arguments: three 2D arrays no wider than `NUMCOLS`, the number of rows and number of columns in each array.
Assume each of the 2D arrays is a matrix, each with the same dimensions(*m* rows by *n* columns).
You are to add the first and second matrix together, storing the result in the third matrix.
Here is an example of two matrices being added to produce the answer on the right:

$$\begin{matrix} 2 & -3 \\ -7 & 5 \end{matrix} \quad + \quad \begin{matrix} 1 & 4 \\ 0 & 2 \end{matrix} \quad = \quad \begin{matrix} 3 & 1 \\ -7 & 7 \end{matrix}$$

```
/* purpose: stores the sum of m1 and m2 to result
 * parameters: int m1[][NUM_COLS], int m2[][NUM_COLS], int result[][NUM_COLS]
 *  int num_rows, number of rows in m1, m2 and result, >=0
 *  int num_cols, number of columns in m1, m2 and result, >=0  and <NUM_COLS
 */
void sum_matrices(int m1[][NUMCOLS], int m2[][NUMCOLS], int result[][NUMCOLS],
                  int num_rows, int num_cols) {
   int row, col, sum;

   for( row = 0; row < num_rows; row++ ) {
      for( col = 0; col < num_cols; col++ ) {
         result[ row ][ col ] = m1[ row ][ col ] + m2[ row ][ col ];
      }
   }
}
```

9. Complete the `main` function that so that it tests the `sum` function you wrote above.

```
#include <stdio.h>
#define NUMCOLS 3

void sum_matrices(int m1[][NUMCOLS], int m2[][NUMCOLS], int result[][NUMCOLS],
                  int num_rows, int num_cols);
void print_array(int array[], int num_elements);

int main( void ) {
    int data1[][3] = {{ 1, 4, -5 },
                      { 5, -2, 6 },
                      { -2, 7, 5 },
                      { 1, 2, 3 }};
    int data2[][3] = {{ 2, 4, 7 },
                      { 3, 5, 2 },
                      { 3, 0, 1 },
                      { 2, 3, 5 }};
    int result[4][3];

    int num_cols = 3;
    int num_rows = 4;
    sum_matrices(data1, data2, result, num_rows, num_cols);

    // print result
    int row;
    for( row = 0; row < num_rows; row++ ) {
        print_array(result[row], num_cols);
    }

    return 0;
}

/* purpose: stores the sum of m1 and m2 to result
 * parameters: int m1[][NUM_COLS], int m2[][NUM_COLS], int result[][NUM_COLS]
 *  int num_rows, number of rows in m1, m2 and result, >=0
 *  int num_cols, number of columns in m1, m2 and result, >=0  and <NUM_COLS
 */
void sum_matrices(int m1[][NUMCOLS], int m2[][NUMCOLS], int result[][NUMCOLS],
                  int num_rows, int num_cols) {
   int row, col;

   for( row = 0; row < num_rows; row++ ) {
      for( col = 0; col < num_cols; col++ ) {
         result[ row ][ col ] = m1[ row ][ col ] + m2[ row ][ col ];
      }
   }
}
/* purpose: prints num_elements in array
 * parameters: int array[],
 *             int num_elements, number of values in array, >=0
 */
void print_array(int array[], int num_elements) {
    int i;

    for(i=0; i<num_elements; i++){
        printf("%d ", array[i]);
    }
    printf("\n");
}
```