Question 1. [10 points] Assume that there is a grid of h rows and w columns of integer values. Also, assume that q is an arbitrary integer. Using pseudo-code, briefly sketch a *sequential* algorithm (i.e., *not* parallel) that will count how many 3x3 blocks of values sum to exactly q.

For example if $q = 7$ and the grid is

w-2

2	0	0	1	0	0
2	0	<u>2</u>	<u>0</u>	0	1
0	<u>2</u>	<u>0</u>	2	<u>2</u>	0
0	0	<u>1</u>	0	<u>1</u>	1
1	0	2	0	<u>1</u>	0
1	0	1	1	1	2

h-2

centers of 3x3 blocks

e.g. 2x2 decomposition for parallel algorithm

then the result of the algorithm should be 8. (The grid locations at the center of each 3x3 block whose sum is 7 are underlined in the grid shown above.)

```
local_sum = 0
for (i = 1 ; i < h-1 ; i++) {
    for (j = 1; j < w-1; j++) {
        sum = [sum of 3x3 square surrounding
               cell at row i, column j]
        if ( sum == q) {
            local_sum ++
        }
    }
}
[local-sum is the solution]
```

Question 2. [30 points] Using pseudo-code, sketch an *parallel* algorithm for the problem described in Question 1. Your parallel algorithm should divide the overall h by w grid up into an N (rows) by M (columns) grid of processors, assigning a smaller local grid to each processor. Make sure your pseudo-code shows

- How each processor determines which portion of the overall grid it will work on
- How the local results computed by each processor are combined to form an overall solution

```
proc_row = rank / M                                  Divide up
proc_col = rank % M                                  work
eff_w = w - 2
eff_h = h - 2
    chunk_size_horiz = eff_w / M
    startx = proc_col * chunk_size_horiz
    if (proc_col == M-1) {
        chunk_size_horiz += eff_w % M
    }
    endx = startx + chunk_size_horiz
chunk_size_vert = eff_h / N
starty = proc_row * chunk_size_vert
if (proc_row == N-1) {
        chunk_size_vert += eff_h % N
}
endy = starty + chunk_size_vert
local_sum = 0
for ( i = starty + 1 ; i < endy + 1; i++) {
    for ( j = startx + 1; j < endx + 1; j++) {
        [ same as seq. alg. ]
    }
}
// local_sum is # of 3x3 blocks in
// local process summing to 9
All processes: reduce local sums by adding
to yield a single global sum
```

Question 3. [60 points] Implement your parallel algorithm using MPI. To get started, see the instructions on the exam web page:

```
http://ycpcs.github.io/cs365-spring2015/assign/exam01.html
```

Edit the code in `countblocks.c`. To run the program, use the command

`./runpar` *filename* q N M

where *filename* is an input file, q is the value of the integer q, N is the number of rows of processes, and M is the number of columns of processes.

The output of the program should be

`Result is` *number*

where *number* is the total number of 3x3 blocks whose sum was q.

Some hints and suggestions:

- Code to read the input data into a `Grid` object is provided
- Because the local processes don't modify any data (they just read the already-loaded data), you don't need to allocate a local `Grid` or copy any data into it: each process can use a region of the global `Grid`
- Use the `grid_get_current` function to get values from the global `Grid`
- You can use the `divide_work` function to help each process determine which ranges of rows and columns it should check (but you aren't required to use it)
- Only one process should report the overall result

Two test files are provided. Some tests you can try:

```
./runpar test1.dat 7 2 2
```

The output should be `Result is 8`.

Another test:

```
./runpar test1.dat 8 2 2
```

The output should be `Result is 3`.

Another test:

```
./runpar test2.dat 7 2 2
```

The output should be `Result is 44`.