Programming Project #6

In this assignment we will do various manipulations of vectors, especially 2D vectors represented as 1D vectors.

Description

A question I get asked quite regularly is “what are the coding questions like in technical interviews?” This project is a kind of answer to that question. I looked around and found various tech-interview questions involving vectors and coded them as functions. These are the kinds of questions interviewers might ask you to sketch out on a whiteboard (or even write code for). The functions are not very related, nor are they each individually very hard. But they are representative of the kinds of things you might be asked to answer on a tech interview (at least if the questions were regarding 2D vectors).

2D to 1D and back again

A 2D vector is a matrix, a representation that has a column and row index for each value. A 2D vector is shown below. Note that the row and column indicies are shown in bold. They are not part of the matrix.

	0	**1**	**2**	**3**	**4**
0	11	12	13	14	15
1	21	22	23	24	25
2	31	32	33	34	35
3	41	42	43	44	45

So the value at row:2, column:3 is 34 (rows and columns both start at 0)

However, there is a fairly easy way to translate between a 2D vector as shown above and a 1D vector. Below is a 1D vector representation of the matrix above with the 1D indicies in bold.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	41	42	43	44	45

The question is really, how to translate row:2, column:3 to a 1D vector index and how to translate a 1D vector index back to a row/column format. To do this we have to know the number of columns in the 2D vector. In this particular case, 5 columns.

row = index/5 (**integer division**), where 5 is the number of columns.

column = index%5 (modulus), where 5 is the number of columns

index = row * 5 + column, where 5 is the number of columns

For the example. row:2 column:3 is index (2 * 5) + 3 → 13

index 13 is row=13/5 →2, column 13%5 → 3

Program Specifications

Each of the functions below is doing calculations on a 2D vector using a 1D vector as the underlying parameter that is passed. The parameters are passed as `const` references: references to prevent copying but `const` to prevent modification of the parameter.

function: `ostream& print_vector(const vector<long>&v, ostream& out)`

print the vector to the provided stream, returning that stream. Each element of the vector is separated from the next by a comma, no comma after the last element. You print out to the provide ostream, **not cout**! Look at the Mimir tests for examples.

You return the stream you passed in!!!

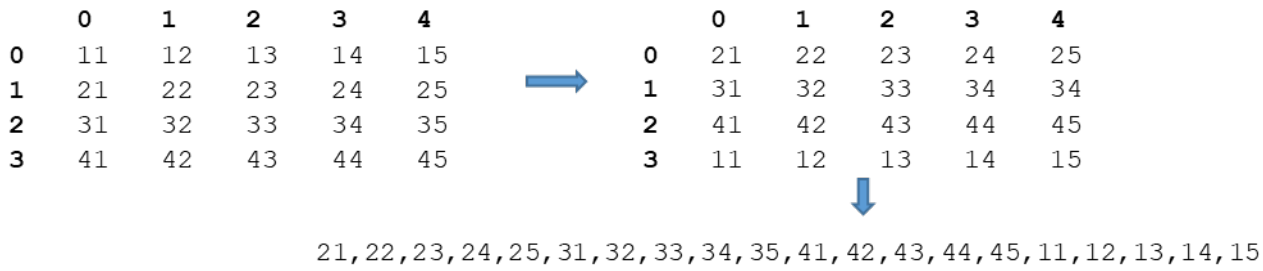
function:
```
long four_corner_sum(const vector<long>& v,
                     int rows, int cols)
```
return the sum of the 4 corners of the provided 1D vector, interpreted as a 2D matrix with provided row and column count. For the matrix above, it would return 11+15+41+45 → 112

Error Check: if there are less than either two rows or two columns (thus no corners), return 0.

function:
```
vector<long> column_order(const vector<long>& v,
                          int rows, int cols)
```
return a new vector that is a reordering of the original 1D vector in column order. Collect each column, first to last, and within the column in row order.

```
  0  1  2  3  4
0 11 12 13 14 15
1 21 22 23 24 25 → 11,21,31,41,12,22,32,42,13,23,33,43,14,24,34,44,15,25,35,45
2 31 32 33 34 35
3 41 42 43 44 45
```

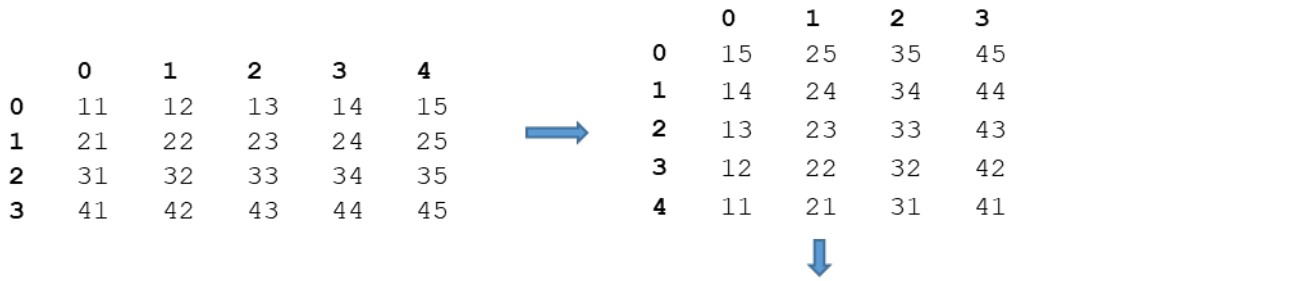
function:
```
vector<long> rotate_rows_up(const vector<long>& v,
                            int rows, int cols)
```
return a new vector that is a rotation of the 2D matrix rows up by one. Here, the 2nd row becomes the 1st, the 3rd the 2nd etc. and the 1st row becomes the last.



Error Check: If there are less than two rows, return an empty vector.

function: `vector<long> matrix_ccw_rotate(const vector<long>& v, int rows, int cols)`

return a new 1D vector that is a rotation of the entire 2D matrix counter-clockwise 90 degrees.



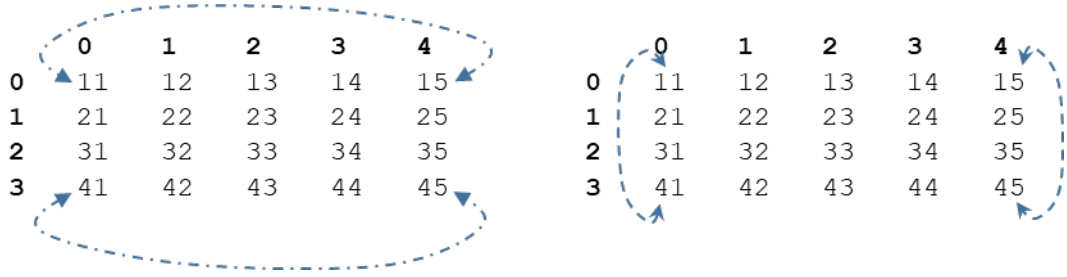
function: `long max_column_diff(const vector<long>& v, int rows, int cols)`

Take the sum of each column. Given those column sums find the maximum difference between any of the two column sums in the matrix. Return the max difference. For our example, the max diff is between column 0 (sum 104) and column 4 (sum 120) → 16.

function: `long trapped_vals(const vector<long>& v, int rows, int cols)`

We are looking to see if any of **values** in the 2D matrix have 4 neighbors (up, down, left, right) that are **all greater** than that value. If so, the value is **trapped**. We return the number of trapped values.

For this, we assume that the 2D matrix displays "wrap-around" behavior. Easier to show than explain.



The value at (0,0), index 0, is 11. We are checking the 4 neighbors (up, down, left, right) of that value to see if they are all greater than 11. The value to the right is (0,1), index 1, is 12. The value below at (1,0), index 5, is 21. The value to the left, well there is no value to the left. But we assume the rows **wrap around**. Thus the a column left of (0,0), index 0 is (0,4), index 4, value 15. The value above, well there is no value above. But we assume the columns also wrap around. Thus a row up from (0,0), index 0, is (3,0) index 15, value 41.

11 at (0,0), index 0 is a trapped value. It is the only trapped value in this example.

Deliverables

You will turn in one file: `proj06_functions.cpp`. We provide you only with `proj06_functions.h` , you must write your own main to test your functions. Mimir can test the individual functions without a main program but it's a good idea for **you** to test your own code with a main, perhaps in the manner that we did previously.

Remember to include your section, the date, project number and comments and you ***do not provide*** main.cpp. If you turn in a main with your code Mimir will not be able to grade you.

1. Please be sure to use the specified file names
2. Always a good idea to save a copy of your file in your H: drive on EGR.
3. Submit to Mimir as always. There will be a mix of visible and not-visible cases.

Assignment Notes

1. You turn in the functions only. To test against your own main you can write a separate file with the main and then compile the two files at the same time. See the lab and videos for examples.
2. If figuring out the row,column to index and back is difficult, perhaps a function or two would be helpful!