You shall submit a zipped, **and only zipped**, archive of your homework directory, hw3. The directory shall contain, at a minimum, the files `src/find_sum.cpp` and `inc/find_sum.h`. Name the archive submission file hw3.zip

I will use my own makefile to compile and link to your `src/find_sum.cpp` and `inc/find_sum.h` files. You must submit, at least, these two files.

Description

Consider word puzzles in which you search for words hidden in a scramble of letters. Consider an algorithmic solution to this. You are tasked with solving a different problem that essentially removes repetitive code:

> Determine whether you can create one or more summations given a goal value from the addition of the sub-rows, sub-columns, or sub-diagonals of a matrix of integers.

You will provide me with a library containing (at least) the function `FindSum` as described in the file `find_sum.h`. You should read the header file as well as my test file to understand what is expected from the library. Do note that the preconditions keep you from any unpleasantness due to array bounds and the like.

I have provided you a basic test application which you can use to ensure that your code is, at least partially, correct. I would suggest a more rigorous testing scheme to ensure that your methods handle any edge cases.

To call the program with the provided test files, reference the makefile rule for test-find-sum. The tests correspond to

- ./test-find-sum dat/row_matrix.txt 0 : row summations
- ./test-find-sum dat/col_matrix.txt 1 : column summations
- ./test-find-sum dat/asc_matrix.txt 2 : diagonal ascending summations
- ./test-find-sum dat/dsc_matrix.txt 3 : diagonal descending summations
- ./test-find-sum dat/matrix.txt 4 : no summations
- ./test-find-sum dat/matrix.txt 5 : single digit summations,
- ./test-find-sum dat/large_matrix.txt 6 : large number of summations,
- test-find-sum-memory which providing a memory leak-check, and
- test-find-sum-style runs cpplint with correct parameters

Points

You will receive points if your `FindSum` library:

i. compiles (1 point)

ii. finds row summations (1.5 points)

iii. finds column summations (1.5 points)

iv. finds ascending diagonal summations (1.5 points)

v finds descending diagonal summations (1.5 points)

vi determines no summations found (must complete all 4 summations correctly) (0.25 points)

vii handles a large number of summations (must handle ii–v above) (0.25 points)

viii finds summations of length one (must complete all 4 summations correctly) (0.25 points)

ix has no memory leaks (must handle cases ii–viii above) (0.25 points)

x has correct styling (2 points)

The End.