CME213, Spring 2022
Parallel Computing
Julian Cooper

Homework 1
Due Friday, April 8th via GradeScope

Problem 1: Implement the `Matrix` and `MatrixSymmetric` class in the given `Matrix.hpp`.

Idea: only store diagonal and one side (upper or lower triangle) of the symmetric matrix data to avoid duplication and meet required memory constraints: $n(n+1)/2 + \mathrm{O}(1)$.

Submitted code passes all tests in `main_q1.cpp`. See output below.

```
$ make main_q1
g++ -std=c++11 -g -Wall -Wextra -pedantic main_q1.cpp -o main_q1

$ ./main_q1
Default constructor test passed.
Empty matrix test passed.
Negative size constructor test passed.
L0Norm test passed.
Initialization and retrieval tests passed.
Out of bounds test passed.
Stream operator test passed.
```

Problem 2: Complete the code given in `main_q2.cpp` and demonstrate how we might append different types of matricies to the same `std::vector`.

Idea: setup polymorphism relationship between the `Matrix` base class and the `SparseMatrix` and `ToeplitzMatrix` derived classes.

This allows us to call the same `repr()` method on different matrix types and our program will sort out which derived string representation function to execute.

Our example with two matrix types demonstrates this concept. See console output below.

```
$ make main_q2
g++ -std=c++11 -g -Wall -Wextra -pedantic main_q2.cpp -o main_q2

$ ./main_q2
sparse
toeplitz
```

Problem 3: Complete the code given in `main_q3.cpp` and demonstrate that function `count_range()` efficiently queries the number of entries in a given set.

Idea: use in-built lower and upper bound methods from `set` class to find iterators pointing to the first element 'not less than' `lb` and the first element 'greater than' `ub`.

We then can compute the "distance" between these two iterators to find the count of entries contained by our set within range [lb, ub].

Console outputs below.

```
$ make main_q3
g++ -std=c++11 -g -Wall -Wextra -pedantic main_q3.cpp -o main_q3

$ ./main_q3
Range test passed.
Count test passed.
Number of elements in range [-1, 1]: 6843 (est. = 6830)
```

Problem 4: Solve the following involving the c++ standard library. You are not allowed to use any loops outside of testing.

- **4a. Implement DAXPY** Since our DAXPY function takes const references vector arguments x and y, I chose to use `std::transform` from the standard library to avoid inplace operations. I then implement $ax + y$ logic using a short lambda function.
- **4b. Compute student grades.** Idea: use `std::all_of` to check whether all students in our vector input argument passed their course. We can incorporate the logic associated with 'passing' in the predicate, which in this case can be a short lambda function.
- **4c. Sort with odd first.** Our sort criteria for this problem is non-standard which pushes us to implement a custom comparison functor (struct with operator()) to use in the `std::sort` algorithm.
- **4d. Sort linked list.** Not all sorts work on linked lists. For this problem we need to use the `std::list<T>::sort` in-built list sort method and incorporate a custom comparison functor as we did in part (C). In this case, our comparison functor needs to interpret public member attributes from the `SparseMatrixCoordinate` class to resolve its boolean operation.

Console logs from 4a-d.

```
$ make main_q4
g++ -std=c++11 -g -Wall -Wextra -pedantic main_q4.cpp -o main_q4

$ ./main_q4
Q4a: PASSED
```

```
Q4b: PASSED
Q4b: PASSED
Q4b: PASSED
Q4b: PASSED
Q4c: PASSED
Q4d empty list: PASSED
Q4d: PASSED
```