Exercise 1 STL Algorithms

1. (Nonmodifying Algorithms)
We include a number of exercises that process STL containers and produce some output, usually a number or a data structure. For convenience, use STL vectors as the input container and note that are several possible solutions to a given problem, some more efficient and/or readable than others. It is up to you to decide how to design the algorithm and the structure/type of the output.

a) Count the frequency of each value in a container. For example, for {1,2,4,5,4,1} we get the output {{1,2}, {2,1}, {4,2}, {5,1}} as pairs of {value, frequency}. There are different ways to model the output depending on your requirements.
b) Write a function to compute the minimum, maximum, sum and average of the values of the elements in a container with numeric values. The output is a tuple with four elements.
c) Consider the container S = {1,2,-3,4,5,5,5,6}. Use an STL algorithm to find the first occurrence of the value.

2. Now use the following functions:
std::bind2nd()
std::bind()
Lambda expression

to find the position of the first even number in S.

d) Consider the container S = {1,2,5,5,-3,4,5,5,5,6,3,4,5}.
Determine how to do the following:
- Return the position of the first three consecutive elements having the value 5.
- Return the position of the first element of the first subrange matching {3,4,5}.
- Return the position of the first element of the last subrange matching {3,4,5}.

e) Consider the container S = {1,2,5,5,-3,4,5,5,5,6,3,4,5}. Find the first element in S that has a value equal to the value of the following element.
f) Consider the container S = {1,2,5,5,-3,4,5,5,5,6,3,4,5} and S1 = {1,2,5}. Determine whether the elements in S1 are equal to the elements in S.

3. (Which Style to use?)
Some STL algorithms use *unary* and *binary predicates*. Both predicate types return a `bool`. A *unary predicate* has one input argument while a *binary predicate* has two input arguments. We can model these predicates and other kinds of functions in a number of ways:
- User-defined function objects.
- Predefined STL function objects (for example, `std::multiplies<T>()`).
- Using lambda functions (possibly with captured variables).

Answer the following questions:
a) Compare these three solutions with regard to quality issues such as readability, understandability and maintainability.
b) Consider transforming a vector of integers into a set of integers. Only those elements whose absolute value is strictly greater than a given *threshold value* should appear in the destination. An example is the vector {1,2,1,4,5,5,-1}. If the threshold value is 2 then the output set will be {4,5}. Implement this problem using the three bespoke methods above.
c) Having developed and debugged the code in part b) review the three solutions from the perspective of understandability, maintainability and efficiency.

4. (Lists versus Vectors)
Lists and vectors are similar in structure but it is important to know the context in which to use each one. Determine whether it is better to use a list, vector or deque in the following cases:

a) Iterating in a container from the beginning to the end and calculating the average of all its elements.
b) Moving to a given (integral) position in the container.
c) Appending and removing elements at the beginning and end of the container.
d) Finding the position in the container corresponding to an element with a given value.
e) Using the container as a building block for creating a container to model matrices.

Determine the complexity (see section 1.3) for each of the above operations. Consider using a map or an unordered map. Would they help in improving performance?