Assignment 2: Introduction to Templates

C++ Programming Course, Winter Term 2016

2-0: Prerequisites

2-0-1: Iterator Concepts

- Make yourself familiar with the `Iterator` concepts in the STL: http://en.cppreference.com/w/cpp/concept/Iterator
- What are the differences between the concepts `ForwardIterator` and `RandomAccessIterator`?
- What are the differences between the concepts `InputIterator` and `OutputIterator`?

2-0-2: Sequence Container Concept

Sequence containers implement data structures which can be accessed sequentially. Methods `begin()` and `end()` define the iteration space of the container elements.

- Make yourself familiar with *Sequence Container* concept defined in the STL: http://en.cppreference.com/w/cpp/concept/SequenceContainer

Excerpt:

Type	Synopsis
`typename value_type`	the container's element type `T`
`typename iterator`	iterator type referencing a container element
`typename const_iterator`	typically defined as `const iterator`
`typename reference`	type definition for `value_type &`
`typename const_reference`	type definition for `const value_type &`

Signature		Synopsis
`iterator begin()`		iterator referencing the first element in the container or `end()` if container is empty
`const_iterator begin()`	`const`	const iterator referencing the first element in the container or `end()` if container is empty
`iterator end()`		iterator referencing past the final element in the container
`const_iterator end()`	`const`	const iterator referencing past the final element in the container
`size_type size()`	`const`	number of elements in the container, same as `end() - begin()`

2-1: The Measurements<T> Class Template

Assuming you run a series of benchmarks, each returning a measurement. At the end of the test series, the mean, median, standard deviation (sigma) and variance should be printed.

Implement the class template `Measurements<T>` representing a sequence container that allows to collect measurement data as single values and provides methods to obtain the mean, median, standard deviation and variance of the container elements.

Measurements Container Concept

In addition to the Sequence Container Concept:

Signature		Synopsis
`T median()`	`const`	returns the median of the elements in the container or `0` if the container is empty.
`double mean()`	`const`	returns the mean of the elements in the container
`double variance()`	`const`	returns the population variance of the elements in the container
`double sigma()`	`const`	returns the standard deviation of the elements in the container

Example:

```
Measurements<int> m1;
m1.insert(10);
m1.insert(34);
```

```
m1.size(); // = 2
Measurements<double> m2;
std::vector<double> v({ 36, 37, 10 });
m2.insert(v.begin(), v.end());

m1.insert(m2.begin(), m2.end())

m1.size(); // = 5

int    median = m1.median();
double mean   = m1.mean();
double sdev   = m1.sigma();
double var    = m1.variance();
```

Define a class template `Measurements<T>` that satisfies the Sequence Container concept (http://en.cppreference.com/w/cpp/concept/SequenceContainer) and the Measurements Container concept defined above.

You may ignore the `emplace` methods for now.

The solution uses `std::vector` as a starting point, but you may use any underlying data structure in your implementation of `cpppc::Measurements<T>`.

2-X: Improve Efficiency

- Refactor your implementation of `Measurements<T>` such that all methods in the *Measurements* concept maintain constant computational complexity $O(c)$
- There are arithmetic solutions, possibly at the cost of numeric stability, and approaches focusing on the underlying data structure

Hints

- Just for fun, have a look at https://en.wikipedia.org/wiki/Standard_deviation#Rapid_calculation_methods
- Review trustworthy (!) references for **multi-index containers**
- DrDobbs is a very trustworthy reference