Assignment 3: Iterators, Function Templates

C++ Programming Course, Winter Term 2016

3-1: Algorithms / Function Templates

Your implementations should be a combination of functions provided by the STL and must not contain explicit loops like `for` or `while`.

You may use any algorithm interface defined in the C++11 standard. For example, you may use `std::minmax` (C++11) in your implementations, but not `std::for_each_n` (C++17).

References:

- Talk by Marshall Clow: "STL Algorithms - why you should use them, and how to write your own" https://www.youtube.com/watch?v=h4Jl1fk3MkQ
- C++ Algorithm Library: http://en.cppreference.com/w/cpp/algorithm
- C++ Iterator Library: http://en.cppreference.com/w/cpp/header/iterator

3-1-1: find_mean_rep

Implement a function template `cpppc::find_mean_rep` which accepts a range specified by two iterators in the `InputIterator` category and returns an iterator to the element in the range that is closest to the mean of the range.

Your implementation should be automatically more efficient for random access iterator ranges without specialization.

Function interface:

```
template <typename InputIter>
InputIter find_mean_rep(InputIter first, InputIter last);
```

Example:

```
std::vector<int> v { 1, 2, 3, 30, 40, 50 }; // mean: 21
auto closest_to_mean_it = cpppc::find_mean_rep(v.begin(), v.end());
// -> iterator at index 3 (|21-30| = 9)
```

3-1-2: histogram (RandomAccessIterator)

Implement a function template `cpppc::histogram` which accepts a range specified by two iterators in the `InputIterator` category and replaces each value by the number of its occurrences in the range. Values are unique in the result histogram and ordered by their first occurrence in the range. The function returns an iterator past the final element in the histogram.

Only integer values (int, long, size_t, ...) have to be supported.

Function interface:

```
template <typename RAIt>
RAIt histogram(RAIt first, RAIt last);
```

Example:

```
std::vector<int> v { 1, 5, 5, 3, 4, 1, 5, 7 };
auto hist_end = cpppc::histogram(v.begin(), v.end());
// -> 1: 2 occurences
//    5: 3 occurences
//    5: skipped
//    ...
// -> { 2, 3, 1, 1, 1 }  <- occurrences
//       ^  ^  ^  ^  ^
//       |  |  |  |  |
//       1  5  3  4  7    <- value
//
// -> returns iterator at position 5
```

3-2: Custom Container / Iterator

Implement a sparse array (see https://en.wikipedia.org/wiki/Sparse_array) container that satisfies the `std::array` interface.

A sparse array is a usually large array with most elements set to a default value like 0. Instead of allocating memory for all values, only non-default values are actually allocated.

Different from vectors, arrays are static containers so their size does not change after instantiation. The values of its elements can be iterated and changed, but elements cannot be added or removed.

References:

- std::array in the C++ Standard Template Library: http://en.cppreference.com/w/cpp/container/array

```
int default = 0;
cpppc::sparse_array<int, 10000> sa(default);
// no actual values in `sa` yet, all set to default
size_t sa_size = sa.size(); // -> 10000
assert(sa[345] == 0);       // returns `default` if no value
                            // set at position

sa[230] = 23;
assert(sa[230] == 23);

sa[420] = 42;
// two values stored in `sa`

// Must be compatible with STL algorithms:

auto found_42 = std::find(sa.begin(), sa.end(), 42);
// -> iterator at 'virtual' position 420 (the second 'real' value)
```

Notes:

- Your implementation has to detect assignment of a value reference. Try a temporary proxy: https://en.wikibooks.org/wiki/More_C++_Idioms/Temporary_Proxy.
- Obviously, the `sparse_array` cannot provide direct access to its underlying array (as it doesn't exist) so method `data()` is not provided.

3-X: Challenges

3-X-1: Skip List

Implement a class template `cpppc::map` that satisfies the `std::map` container interface based on a Skip List data structure.

References:

- Specification of std::map: http://en.cppreference.com/w/cpp/container/map
- *Skip Lists: A Probabilistic Alternative to Balanced Trees*. W. Pugh, 1990. http://cpppc.pub.lab.nm.ifi.lmu.de/docs/skiplist.pdf