unordered_map homework

Part 1 (implementation of some std::unordered_map methods)

For this part of the homework there are no test cases written. You should solve the problems and write some good tests too by yourself. Try to test mostly corner cases.

1. Add a growing mechanism:
   1.1 Update the `mapped_type& operator[](const Key& k)` function to grow (rehash) the map when `load_factor >= max_load_factor`.
   1.2 Implement the `rehash` method.
   Sets the number of buckets to count and rehashes the container, i.e. puts the elements into appropriate buckets considering that the total number of buckets has changed. If the new number of buckets makes load factor more than maximum load factor (`count < size() / max_load_factor()`), then the new number of buckets is at least `size() / max_load_factor()`
2. implement the `void max_load_factor(float ml)` function to change the default load_factor.
3. Implement the `bool operator==(const _unordered_map& other)` function.
4. Implement the `void reserve( size_type count );` function.
   Sets the number of buckets to the number needed to accommodate at least count elements without exceeding maximum load factor and rehashes the container, i.e. puts the elements into appropriate buckets considering that total number of buckets has changed. Effectively calls `rehash(std::ceil(count / max_load_factor()))`.
5. Implement the `size_type bucket( const Key& key ) const` function.
   Returns the index of the bucket for key key. Elements (if any) with keys equivalent to key are always found in this bucket.
6. Implement the `size_type bucket_size( size_type n ) const` function.
   Returns the number of elements in the bucket with index n.
7. Implement the `size_type max_bucket_count() const` function.
   Returns the maximum number of buckets the container is able to hold due to system or library implementation limitations

Part 2 (hashing)

1. Implement multiplicative hashing from Knuth's book with the most efficient way, pass it to our `_unordered_map` object and add some tests. Compare the `load_factor` with the default hash: `std::hash`.
2. Design some good hash functions for strings and again do some tests with load_factor comparing it with the default hash.

Part 3 (Algorithmic Problems)
//TODO