Programming in C^{++}, Exercise List 7

Deadline: 28.04.2017

In this exercise, we study `std::map< >` and `std::unordered_map< >`. They have similar functionality: Each of the two versions of `map<X,Y>` implements a table of elements (x, y) with $x \in X$ and $y \in Y$, in such a way that y can be efficiently looked up, when x is known. One could also say that `map<X,Y>` implements a lookup table from X to Y.

The difference between `std::map<X,Y>` and `std::unordered_map<X,Y>` is the mechanism that is used for lookup: `std::map< >` uses a search tree, so that it requires an order on type X. `std::unordered_map< >` is based on hashing, so it needs a hash function and an equality function on type X.

1. Write a function

   ```
   std::map< std::string, unsigned int > frequencytable(
             const std::vector< std::string > & text )
   ```

   that constructs a table of frequencies of the words in `text`.

   Inserting into a map can be tricky when Y has no default constructor, but in this task you can simply use `[ ]`.

2. Write a function

   ```
   std::ostream&
   operator << ( std::ostream& stream,
                 const std::map< std::string, unsigned int > & freq )
   ```

   that prints the frequency table. Use a **range-for**.

   Note that in real code, frequencytable should always be made a separate class, because if one defines `operator <<` on `std::map< std::string, unsigned int >`, printing it as a frequency table, one has no possibility to use `std::map< std::string, unsigned int >` for something else anymore.

3. `std::map< >` uses by default the order `<` on `std::string`. We want the frequence table to be case insenstive. Try for example:

   ```
         std::cout << frequencytable( std::vector< std::string >
            { "AA", "aA", "Aa", "this", "THIS" } );
   ```

In order to solve this problem, we will have to provide our own comparator. Define a class

```
struct case_insensitive_cmp
{
   bool operator( ) ( const std::string& s1, const std::string& s2 ) const;
      // Return true if s1 < s2, ignoring case of the letters.
};
```

Class `case_insensitive_cmp` has only one constructor, namely its default constructor. Test it for example by

```
case_insensitive_cmp c;
std::cout << c( "a", "A" ) << c( "a","b" ) << c( "A", "b" ) << "\n";
```

There is no `==`-operator. `std::map` will assume that two objects `s1`,`s2` are equal when both `c(s1,s2)` and `(s2,s1)` are false.

Write `bool operator( )` in a reasonable fashion! Making a lower case copy of the string, and using `<` is not reasonable.

4. Once you have finished the `case_insensitive_cmp` class, you can do one of the following things, dependent of your level of eagerness:

   - Simply replace `std::map< std::string, unsigned int >` by `std::map< std::string, unsigned int, case_insensitive_cmp >`, in everything that you wrote before, and comparison should now be case-insensitive.
   - Make `operator <<` and `frequencytable` polymorphic: Write:

     ```
     template< typename C = std::less< std::string >>
     std::map< std::string, unsigned int, C >
     frequencytable( const std::vector< std::string > & text )
     ```

     This makes `frequencytable` polymorphic. Parameter `C` is the comparator, which by default is `std::less< std::string >`.

     Calling `frequencytable( test )` will produce a frequency table using default `std::less< std::string >` which is just $<$ on `std::string`.

     Calling `frequencytable< case_insensitive_cmp >( test )` will produce a case insensitive frequency table.

     Replace the print function by:

     ```
     template< typename C >
     std::ostream& operator << ( std::ostream& out,
                   const std::map< std::string, unsigned int, C > & m )
     ```

     There is no need to give a default value for C, because it will be derived from the type of the argument.

5. Now we want to write the same functions with `std::unordered_map`. If we will do nothing, comparison will also be case sensitive here, so we need to create a case-insensitive hash function, and a case-insensitive equality function. They work in the same way as the `case_insensitive_cmp` object:

   ```
   struct case_insensitive_hash
   {
      size_t operator ( ) ( const std::string& s ) const;
   };

   struct case_insensitive_equality
   {
      bool operator ( ) ( const std::string& s1,
                          const std::string& s2 ) const;
   };

   case_insensitive_hash h;
   std::cout << h( "xxx" ) << " " << h( "XXX" ) << "\n";
   std::cout << h( "Abc" ) << " " << h( "abC" ) << "\n";
      // Hash value should be case insensitive.

   case_insensitive_equality e;
      std::cout << e( "xxx", "XXX" ) << "\n";
         // Prints '1'.
   ```

6. If everything went well, you can now easily write

   ```
   std::unordered_map< std::string, unsigned int,
                       case_insensitive_hash, case_insensitive_equality >
   hashed_frequncytable( const std::vector< std::string > & text ),
   ```

   or

   ```
   template< typename H = std::hash< std::string >,
             typename E = std::equal_to< std::string >>
   std::unordered_map< std::string, unsigned int, H, E >
   hashed_frequencytable( const std::vector<std::string> & text );
   ```

7. Download the first book of ‘Confessiones’ from `http://www9.georgetown.edu/faculty/jod/latinconf/latinconf.html`. Using the function

   ```
      std::vector< std::string> readfile( const std::string& name )
   ```

   that was written in the previous task, to make a frequency table of the words in the first book. You can use `map` or `unordered_map`.

How often does 'magnus' occur? And 'hominum' and 'memoria'?

What is the most frequent word? There is no efficient way to find it, you have to traverse the complete map. Use a `const_iterator`, and use `end( )` for the undefined value.