Programming in C^{++}, Exercise List 6

Deadline: 01/03.10.2018 (the day of your assigned lab)

In this exercise, we study `std::map<X,Y>` and `std::unordered_map<X,Y>`. They have similar functionality: Each of the two versions 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 red black tree, so that it requires an order on type X. `std::unordered_map< >` uses 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 it is easy, because 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 the default order `<` on `std::string`. We want the frequency table to be case insensitive. 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 order. 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 on

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

There is no `==`-operator. `std::map` will simply 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 strings, and using `<` is not reasonable!

4. Once you have finished the `case_insensitive_cmp` class, you can replace `std::map< std::string, unsigned int >` by `std::map< std::string, unsigned int, case_insensitive_cmp >`, in everything that you wrote before, and now comparison is case insensitive.

5. Now we want to write the same functions with `std::unordered_map`. If we 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_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 you wrote in the previous task, make a frequency table of the words in the first book. You can either use `map` or `unordered_map`.

How often does the word '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. Write a function that does it. Use a `const_iterator`, and use `end( )` for the undefined value.