Course C^{++}, Exercise List 7

11.04.2013

In this exercise, we study `std::map< >` and `std::unordered_map< >`. They have similar functionality: Each version 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 looked up efficiently, 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< >` is based on red-black trees, and it requires an order `<` on X. `std::unordered_map< >` is based on hashing, so it needs a hash function and an equality function on 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 subtle, but in this task you can simply use `[ ]`. In a later exercise, we will treat `[ ]` in more detail, because it has some problems with constness and default constructors in Y.

2. Write a function

   ```
   std::ostream& operator << ( std::ostream& stream,
                               const std::vector< std::string > & vect )
   ```

   that prints the frequency table. Use a `map::const_iterator`. (You may use it by **auto**, but make sure that you use an iterator.

3. `std::map< >` uses by default the order `<` on `std::string`. This is not quite what we want, because `<` is case sensitive. Try for example:

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

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

```
struct cmp
{
   bool operator( ) ( const std::string& s1, const std::string& s2 ) const;
      // Return true if s1 < s2, ignoring case of the letters.
      // (Equality is tested by making two comparisons,
      //  s1 < s2 and s2 < s1. If they both fail, it is assumed  that
      //  the strings are equal.
};
```

`cmp` has only one constructor, which is its default constructor. You can write for example:

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

There is no `=`-operator. The map will assume that two objects are equal if both `c(s1,s2)` and `(s2,s1)` are false. One you have finished the **cmp** class, you can replace `std::map< std::string, unsigned int >` by `std::map< std::string, unsigned int, cmp >`, and everything should work fine.

4. Now we want to write the same function with `std::unordered_map`. There is the same problem with case distinction, so we need to create a case-insensitive hash function, and a case-insensitive equality function. They work in the same way as the **cmp** object:

```
struct hash
{
   unsigned int operator ( ) ( const std::string& s ) const
};

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

hash h;
std::cout << h( "xxx" ) << " " << h( "XXX" ) << "\n";
   // Prints twice the same number.

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

std::unordered_map< std::string, unsigned int, hash, equals > freq = ..
```

5. 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, make a frequency table of the words in the first book.

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.