Course C^{++}, Exercise List 6

08.04.2014

Topic of this exercise are `std::list< >` and `std::vector< >`.

1. Write a function

```
#include <fstream>
#include <vector>
#include <string>

std::vector< std::string> readfile( const std::string& name )
{
   std::ifstream input( name. c_str( ));

   while( input. good( ) && !input. eof( ))
   {
      int c = input. get( );
      ....
   }

}
```

   The function must read the complete text file **name**, and return its contents as a vector of strings. The strings are the words occurring in **name**. Assume that words are separated by whitespace (tab, space or return). Whitespace of more than one character should not result in empty words.

   Find a text file that is not too small to test it on. (I am sure, there is a text file somewhere in the internet.)

   Use

```
std::ostream& operator << ( std::ostream& stream,
                            const std::vector< std::string > & vect )
{
   for( std::vector< std::string > :: const_iterator
           p = vect. begin( );
           p != vect. end( );
           ++ p )
```

```
   {
      stream << *p << "\n";
   }
   return stream;
}
```

2. Consider the sorting functions:

```
void sort( std::vector< std::string > & v )
{
   for( unsigned int j = 0; j < v. size( ); ++ j )
      for( unsigned int i = 0; i < j; ++ i )
      {
         if( v[i] > v[j] )
         {
            std::string s = v[i];
            v[i] = v[j];
            v[j] = s;
         }
      }
}
```

and

```
void sort( std::vector< std::string > & v )
{
   for( unsigned int j = 0; j < v. size( ); ++ j )
      for( unsigned int i = 0; i < j; ++ i )
      {
         if( v[i] > v[j] )
         {
            std::string s = std::move( v[i] );
            v[i] = std::move( v[j] );
            v[j] = std::move( s );
         }
      }
}
```

Try to measure a difference in performance. Is there some? (Use input from a big file.)

The easiest way to measure peformance is by using the `time` command. Use `time ./task6`, where `task6` is the time of your program. In order to make sure that the time of reading the file can be neglected, repeat the sorting a couple of times, e.g. 1000.

```
      std::vector< std::string > input = readfile( );
```

```
for( unsigned int i = 0; i < 1000; ++ i )
{
   auto input2 = input;
   sort( input2 );
}
```

You can switch compiler optimization on, by using option `-O` of `gcc`. You can also try to make the strings longer, e.g. by writing

```
std::vector< std::string > input = readfile( );
for( unsigned int i = 0; i < input. size( ); ++ i )
{
   for( unsigned int j = 0; j < 4; ++ j )
      input[i] = input[i] + input[i];
}
```

3. Instead of indexing, it is much nicer in general to use iterators. `std::vector` has two iterator types, `std::vector<X> :: iterator` and `std::vector<X> :: const_iterator`. Since `v` is const, we can only use `const_iterator`.

   You can see in `operator <<` above how iterators are used. Instead of the iterator declaration, one can use `auto` in C^{++}-11. (So one can write `for( auto p = v. begin( ); p != v. end( ); ++ p )`)

   Rewrite the sorting function with iterators.

4. Rewrite the previous functions (reading a file, sorting a list of strings, printing a list of strings) using `std::list< >`. The difference shouldn't be big. When using list iterator, you have to be aware of the fact that `<` doesn't exist for list iterators. Replace it by `!=`.

5. Write a function

   ```
   void removeshortstrings( std::list< std::string > & lst,
                            unsigned int len )
   ```

   that removes all strings that are shorter than `len` from `lst`.

   Don't copy the list, use `erase( iterator it )`.

6. Can you measure a difference in performance between the list version and the vector version? Is there a difference in memory use?