Course C^{++}, Exercise List 6

14.04.2015

This eclectic exercise covers may topics at the same time: Usage of `std::list< >` and `std::vector< >`, file handling, use of namespaces, use of input parameters, and time measuring. Namespaces are a convenient way of avoiding name conflicts in big programs. Our program will be not so big, but we need to get used to using them.

Download the files **listtest.h**, **listtest.cpp**, **vectortest.h**, **vectortest.cpp**, **nr06.cpp** and the **Makefile** from the course homepage.

1. Complete the function

   ```
   void readfile( std::istream& input,
                  std::vector< std::string > & vect )
   ```

   in file **nr06.cpp**. This function must read all words from the inputfile and append them to vect. It should ignore whitespace and interpunction. Repeated whitespace and interpunction must not result in empty words. The function must never produce empty words. Whitespace can be recognized by `isspace( int )` and interpunction can be recognized by `ispunct( int )`.

   `input.good( )` means that the last operation on `input` succeeded.

2. Complete the functions

   ```
   std::ostream&
   operator << std::ostream& , const std::vector< std::string > & );
   std::ostream&
   operator << std::ostream& , const std::list< std::string > & );
   ```

   in files **vectortest.cpp** and **listtest.cpp**. They are not in the namespace, because uniqueness is guaranteed by their type.

   The preferred way of implementing these functions is by using a range-for.

3. Add the following sorting functions to **vectortest.cpp**:

```
#if 0
void vectortest::sort( std::vector< std::string > & v )
{
   for( size_t j = 0; j < v. size( ); ++ j )
      for( size_t i = 0; i < j; ++ i )
      {
         if( v[i] > v[j] )
         {
            std::swap( v[i], v[j] );
            // std::string s = v[i];
            // v[i] = v[j];
            // v[j] = s;
         }
      }
}
#endif
#if 0
void vectortest::sort( std::vector< std::string > & v )
{
   for( auto q = v. begin( ); q != v. end( ); ++ q )
      for( auto p = v. begin( ); p != q; ++ p )
      {
         if( *p > *q )
         {
            std::swap( *p, *q );
            // std::string s = *p;
            // *p = *q;
            // *q = s;
         }
      }
}
#endif
```

In these sorting functions, there are two choices, giving a total number of four possible implementations. Only two possibilities are given. Write the other two.

The choices are: Using `std::swap` or assignments, and using iterators or indexing. You can call `./nr06` with one or more parameters, which are files that it will read from.

4. Systematically measure the performance of the sorting functions on a file that is big enough, e.g. **1.html**. Which one is the best?

   Use compiler optimization `-O3`.

5. It is often told that bubble sort is quadratic in the size of the data being sorted. Can you observe this? The easiest way to to this is by calling

```
./nr06 1.html
./nr06 1.html 1.html
./nr06 1.html 1.html 1.html
```

(If you want, you can try other sorting algorithms, to find out if it is true what people tell about complexity.)

6. There are people who believe that resulting code becomes faster when compiler optimization is turned on.

   Are you able to observe a difference in speed between optimized compilation `-O3` and non-optimized compilation? How much is it?

7. Write similar sorting functions in file **listtest.cpp** for `std::list`. Only two versions are possible, because list has no indexing.

8. Measure the performance of the two sorting functions on `std::list`. Which one is faster?

9. Finally, compare sorting on `std::list` with sorting on `std::vector`. Which is the fastest of all your sorting algorithms?