Laboratory 3: Introduction to Templates in C++

The aims of today's lab are:

- Finish Lab 2 (the `StringInverter` class);
- Introduce the `vector` class of the Standard Template Library (STL);
- Understand the concept of template functions.

Once you are done, start working on the assignment.

Task -1: Last week

Make sure you finish the `StringInverter` class from Lab 2!

Task 0: Using CMake

Same as before, use CMake to generate the project files for XCode, MS VC++, or Makefiles.

Task 1: STL Vector

To store X elements of the same type, C developers would tend to use an array. C++ programmers would favour the `vector` class provided by the STL. The first solution is not practical when X varies and would require dynamic allocation and recopies of the data, which makes it slow and inefficient.

For the first task of the week, you are given one C++ file (`src/TestVector.cpp`), which corresponds to a short program to get familiar with this `vector` class. At the moment it only contains some C code, your task is to improve it.

1. To use the `vector` class, add at the top of the file:

   ```
   #include <vector>
   ```

   The documentation is in `http://www.cplusplus.com/reference/vector/vector/`

2. In the main, create a vector of 50 random double precision floating point numbers between 0.0 and 1.0. To produce random numbers, you can use the function `randd()` that we provide. The header you need is `<cstdlib>` (for `srand`, `rand()` and `time(0)`). You can see an example below:

```
#include <vector>       // std::vector

int main () {
  std::vector<int> myvector1;
  for (int i=0; i<50; ++i) myvector1.push_back(i*10);

  int i = 0;
  std::vector<int> myvector2(50);
  for (std::vector<int>::iterator ite = myvector2.begin();
       ite != myvector2.end();
       ite++)
  {
       *ite = (i++*10);
  }
  return 0;
}
```

Adapt this code to your own problem.

3. Display every element of the vector using a `const_iterator`, a `for` loop, and `std::cout <<`. Remember to visit `cplusplus.com` to access the official C++ documentation, e.g. `http://www.cplusplus.com/reference/vector/vector/begin/`.

4. In the `for` loop you just wrote, replace `.begin()` by `.rbegin()`, and `.end()` by `.rend()`. What happened?

5. Instead of the for loop, use `std::copy` to display every element of the vector. To do so, you need to include another 2 header files: `#include <algorithm>` for `std::copy`, and `#include <iterator>` for `std::ostream_iterator`. In `http://www.cplusplus.com/reference/iterator/ostream_iterator/`, you can see an example.

```
// ostream_iterator example
#include <iostream>     // std::cout
#include <iterator>     // std::ostream_iterator
#include <vector>       // std::vector
#include <algorithm>    // std::copy

int main () {
  std::vector<int> myvector;
  for (int i=1; i<10; ++i) myvector.push_back(i*10);

  std::ostream_iterator<int> out_it (std::cout,", ");
  std::copy ( myvector.begin(), myvector.end(), out_it );

  // Most people would write:
  std::copy ( myvector.begin(),
              myvector.end(),
              std::ostream_iterator<int>( std::cout,", " )
            );

  return 0;
}
```

Adapt this code to your own problem. The main difference is the template argument. You have a vector of `doubles`, in the example it is an array of `ints`.

6. Remove the last N elements of the vector with:

$$N = 10 \times \text{randd}()$$

You can use the method `vector::erase` or `vector::pop_back`.

7. Now, display the smallest and largest values contained in the vector. To do so, use `std::min_element` and `std::max_element` provided in the `<algorithm>` header. Note that these functions return an iterator. Iterators are a bit like pointers. To display the value pointed by an iterator, add a `*` before it, e.g. `*ite`. See `http://en.cppreference.com/w/cpp/algorithm/min_element` for an example.

8. Finally, display the average value. To get the number of elements in the vector, use its `size()` method. To get the sum of all the elements of the vector, use `std::accumulate()` function. See `http://en.cppreference.com/w/cpp/algorithm/accumulate` for an example. It is provided by the `<numeric>` header. The last parameter is: `0` if you are summing integer numbers, `0.0` for double-precision floating-point numbers, or `0.0f` for single-precision floating-point numbers.

Task 2: Create your own template functions

For this task, you are given two C++ files:

1. `include/Utils.h`, a header file with the declarations of some functions.
2. `src/TestUtils.cpp`, a test program to try your template functions.

Modify `include/Utils.h` to convert every function as a template function. To implement every function, you need to write the code directly in the header. In `src/TestUtils.cpp`, test every template function with different data types.