Homework 7

Computer Architecture I ShanghaiTech University
HW6 HW7

Goals

In this assignment, you can finally code in C++ again. Topics covered are:

- Template Metaprogramming.
- Iterator.
- Const-correctness.
- A better understanding of the STL containers.

Setup

Download the following files: vector.hpp and Makefile.

Overview

Your task is to implement a vector in C++ using templates. Its behaviour should be similar to the `std::vector`. But of course the `std::vector` is more complicated - we implement a simplified version instead. `vector.hpp` defines all the classes needed and some of their data structures and methods. Other methods are left for you to declare. For templates, the program that is using them has to always include the implementation as well, because we cannot generate and link to a .o file because the code depends on the template parameter. To keep the code clean it is one convention to still only declare the classes and functions in a `.hpp` file while its implementation goes into a `*-impl.hpp` file which is included at the end of the `.hpp` file.

Implementation Details

Finish the implementation of the class and methods in `vector-impl.hpp`. And you should obey the following rules:

- No standard libraries are allowed to use. Specifically, no `#include` statements are allowed to appear in `vector-impl.hpp`.
- We assume type `_Tp` has default constructor and destructor, and it is both copy constructible and assignable.
- The number of comments should be at least 25% of the non-blank lines. We will check by hand if those comments are valid and in English - failure to comply may lead to a score of 0 for this HW. You can use this Python script, which is the same as that on Autolab, to check your comments.

Iterator

Finish the implementation of `__detail::__iterator<_Tp>`. You have to implement the following methods:

Constructor: **__iterator** ()
Default constructor.
Constructor: **__iterator** (const __iterator & *other*)
Copy constructor.
Constructor: **__iterator** (_Tp * *ptr*)
Initialize the iterator with a pointer.
Destructor: **~__iterator** ()
Destructor.

You should also overload the following operators:

- `=`: copy assignment operator.
- `*`: dereference operator.
- `->`: arrow operator.
- `==`, `!=`: comparison operator.
- `++`, `--`: self increment and decrement operator.
- `+`, `-`: arithmetic operator.
- `+=`, `-=`: compound assignment operator.

These operators should behave similarly to `std::vector<_Tp>::iterator`.

Const Iterator

It is similar to the iterator except for the dereferenced value should be const type (they cannot be modified). All methods for iterator should also be implemented for const iterator. You should also implement one additional method:

Constructor: **__const_iterator** (const __iterator & *other*)
Convert an iterator into a const iterator. Please notice that the reverse conversion (const iterator -> iterator) is not required and you should consider why.

Vector

Finally, implement the vector. Implement the following methods:

Constructor: **vector** ()
Default constructor.
Constructor: **vector** (size_type *size*, const _Tp &*value*)
Create a vector with `size` copies of `value`.
Constructor: **vector** (std::initializer_list<_Tp> l)
Create a vector consists of the elements in the initializer_list.
Destructor: **~vector** ()
Destructor.
Operator: reference **operator[]** (size_type n)
Subscript access to the data contains in the vector.
Method: size_type **size** () const
Returns the size of the vector.
Method: iterator **begin** ()

Returns an iterator to the first element. If the vector is empty, the returned iterator will be equal to `end()`.

Method: iterator **end** ()
Returns an iterator to the element following the last element.

Method: const_iterator **cbegin** () const
Returns a const iterator to the first element. If the vector is empty, the returned iterator will be equal to `cend()`.

Method: const_iterator **cend** () const
Returns a const iterator to the element following the last element.

Method: iterator **insert** (iterator *pos*, const _Tp &*value*)
Method: iterator **insert** (const_iterator *pos*, const _Tp &*value*)
Insert given value into vector before specified iterator, return an iterator that points to the inserted data.

Method: void **push_back** (const _Tp &value)
Add data to the end of the vector.

Method: iterator **erase** (iterator *pos*)
Method: iterator **erase** (const_iterator *pos*)
Remove the element at given position. Returns an iterator to the next element (or end()).

Method: void **pop_back** ()
Remove last element.

Method: inline void **_grow** ()
Grow vector to the size of capacity.

Please note that the copy and move constructors are removed in order to avoid memory issues like double free.

Hints

- You can refer to the GNU C++ Library's implementation. In GNU/Linux, the header file for `std::vector` is placed at `/usr/include/c++/<GCC version>/`.
- Because the compiler will not generate any code if the templates are not used, so it won't complain about errors exist in methods you do not use. It is strongly suggested that you test all your methods before submission. One easy approach is to specialize your templates and compile it.
- The meaning of `_M_size` and `_M_capacity` is identical to HW2. `_M_size` should not be larger than `_M_capacity`.
- Make use of function `_grow()` properly. You may choose your own way to grow the vector.

Submission

Submit `vector-impl.hpp` to Autolab.

Schwertfeger, Sören <*soerensch AT shanghaitech.edu.cn*>
Chundong Wang <*wangchd AT shanghaitech.edu.cn*>
Last modified: 2020-04-19