

C++ 標準庫

體系結構與內核分析

(C++ Standard Library — architecture & sources)

第一講



侯捷

— 侯捷 —

使用一個東西，
卻不明白它的道理，
不高明！



■■■■ 你應具備的基礎

- C++ 基本語法
(包括如何正確使用模板, templates)

■■■■ 我們的目標

- level 0: 淺嘗 C++ 標準庫
- level 1: 深入認識 C++ 標準庫 (胸中自有丘壑)
- level 2: 良好使用 C++ 標準庫
- level 3: 擴充 C++ 標準庫

■■■■ C++ Standard Library vs. Standard Template Library

➤ C++ Standard Library

C++ 標準庫

➤ Standard Template Library

STL, 標準模板庫

標準庫以 header files 形式呈現

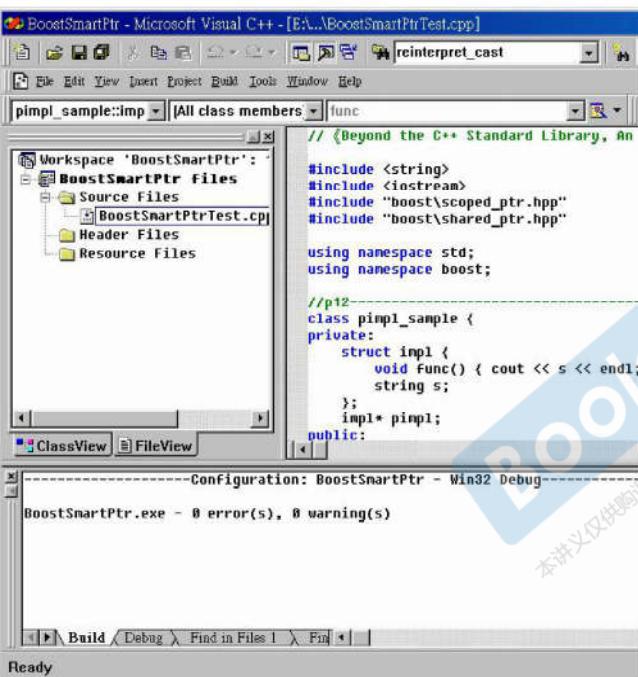
- C++ 標準庫的 header files 不帶副檔名 (.h)，例如 #include <vector>
- 新式 C header files 不帶副檔名 .h，例如 #include <cstdio>
- 舊式 C header files (帶有副檔名 .h) 仍然可用，例如 #include <stdio.h>
- 新式 headers 內的組件封裝於 namespace "std"
 - ➔ using namespace std; or
 - ➔ using std::cout; (for example)
- 舊式 headers 內的組件不封裝於 namespace "std"

```
#include <string>
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;
```

—侯

■■■■ C++ 標準庫, 版本

(Visual C++ 6.0 畫面)



(Dev-C++ 5.11 畫面; with GNU 4.9.2)

```
// 計算上試用各式各樣的 allocators
cout << sizeof(_gnu_cxx::malloc_allocator<int>) << endl;
cout << sizeof(_gnu_cxx::_pool_allocator<int>) << endl;
cout << sizeof(_gnu_cxx::_mt_allocator<int>) << endl;
cout << sizeof(_gnu_cxx::bitmap_allocator<int>) << endl;
cout << sizeof(_gnu_cxx::array_allocator<int>) << endl;
cout << sizeof(_gnu_cxx::debug_allocator<std::all
// cout << sizeof(_gnu_cxx::throw_allocator<int>) << endl;

// 測試 cookie
cout << "sizeof(int)" << sizeof(int) << endl;
cout << "sizeof(double)" << sizeof(double) << endl;

cookie_test(std::allocator<int>(), 1);
cookie_test(_gnu_cxx::malloc_allocator<int>(), 1)
```



重要網頁, CPlusPlus.com

www.cplusplus.com

Search: Go Not logged in register log in

Welcome to cplusplus.com © The C++ Resources Network, 2013

C++

Information
Tutorials
Reference
Articles
Forum

Information
General information about the C++ programming language, including non-technical documents and descriptions:

- Description of the C++ language
- History of the C++ language
- F.A.Q., Frequently Asked Questions

Tutorials
Learn the C++ language from its basics up to its most advanced features.

- C++ Language: Collection of tutorials covering all the features of this versatile and powerful language. Including detailed explanations of pointers, functions, classes and templates, among others...
- more...

Reference
Description of the most important classes, functions and objects of the Standard Language Library, with descriptive fully-functional short programs as examples:

- C library: The popular C library, is also part of the of C++ language library.
- iostream library. The standard C++ library for Input/Output operations.
- String library. Library defining the string class.
- Standard containers. Vectors, lists, maps, sets...
- more...

Articles
User-contributed articles, organized into different categories:

- Algorithms
- Standard library
- C++11
- Windows API
- Other...

You can contribute your own articles!

Forum
Message boards where members can exchange knowledge and comments. Ordered by topics:

C++ Search
Search this website: Search



重要網頁, CppReference.com

The screenshot shows the homepage of CppReference.com. At the top, there is a banner stating "C++ is 30 years old" with the date "October 14, 1985". Below the banner, the page title "C++ reference" is displayed, along with links for "C++98, C++03, C++11, C++14". The main navigation menu on the left includes sections for "Language", "Headers", "Library concepts", "Utilities library", and "Technical specifications". The "Language" section lists "ASCII chart", "Compiler support", "Preprocessor", "Keywords", "Operator precedence", "Escape sequences", and "Fundamental types". The "Headers" section lists "Type support", "Dynamic memory management", "Error handling", "Program Utilities", "Date and time", "bitset", "Function objects", "pair - tuple (C++11)", and "integer_sequence (C++14)". The "Library concepts" section lists "Containers library", "Algorithms library", "Iterators library", "Numerics library", and "Technical specifications". The "Utilities library" section lists "Strings library", "Input/output library", and "Localizations library". The "Technical specifications" section lists "Standard library extensions (library fundamentals TS)" and "Standard library extensions v2 (library fundamentals TS v2)".



重要網頁, [gcc.gnu.org](https://gcc.gnu.org/onlinedocs/gcc-4.6.3/libstdc++/api/a00464.html)

libstdc++

https://gcc.gnu.org/onlinedocs/gcc-4.6.3/libstdc++/api/a00464.html

std::deque Class Reference
Sequences

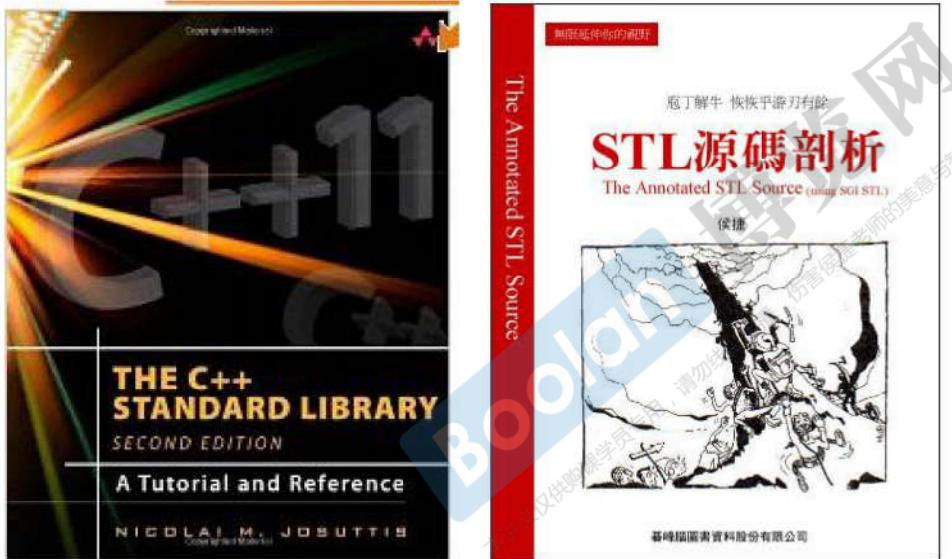
Inheritance diagram for std::deque:

```
graph TD; std::Deque_base["std::Deque_base< _Tp, _Alloc >"] --> std::Deque_base["std::Deque_base< _Tp, _Alloc >"]; std::Deque_base --> std::deque["std::deque"]
```

Public Types

- typedef _Alloc allocator_type
- typedef _Base::const_iterator const_iterator
- typedef _Tp_alloc_type::const_pointer const_pointer
- typedef _Tp_alloc_type::const_reference const_reference
- typedef std::reverse_iterator< const_iterator > const_reverse_iterator
- typedef ptrdiff_t difference_type
- typedef _Base::iterator iterator
- typedef _Tp_alloc_type::pointer pointer
- typedef _Tp_alloc_type::reference reference

■■■■ Bibliography (書目誌)



— 侯捷 —

10

■■■ 你將獲得的代碼

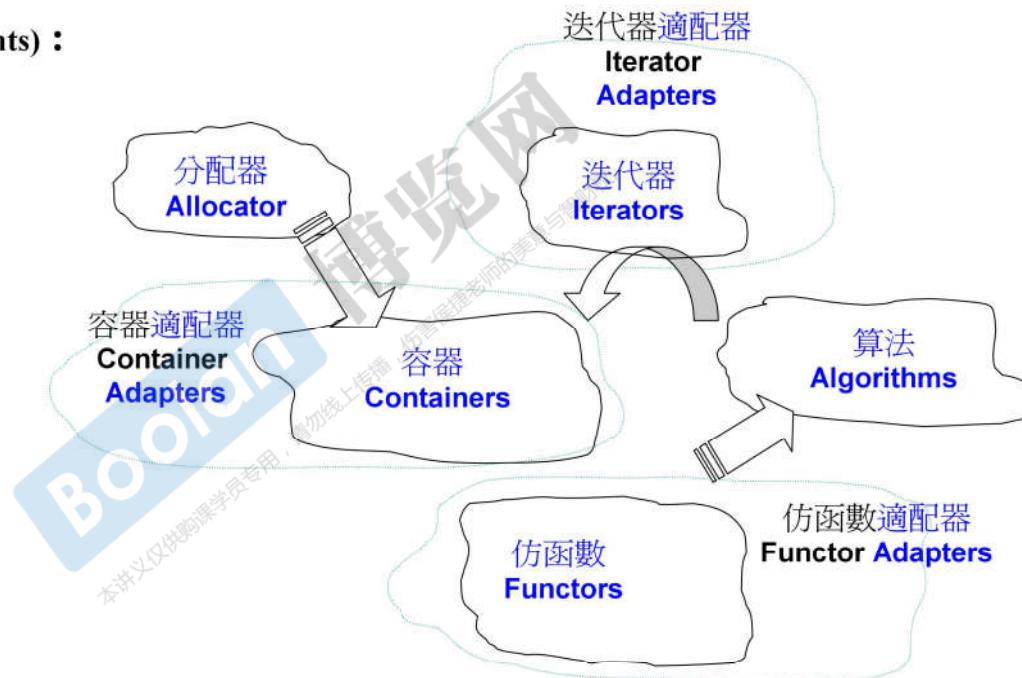
Test-STL.cpp

內有對 C++ 標準庫的各種測試

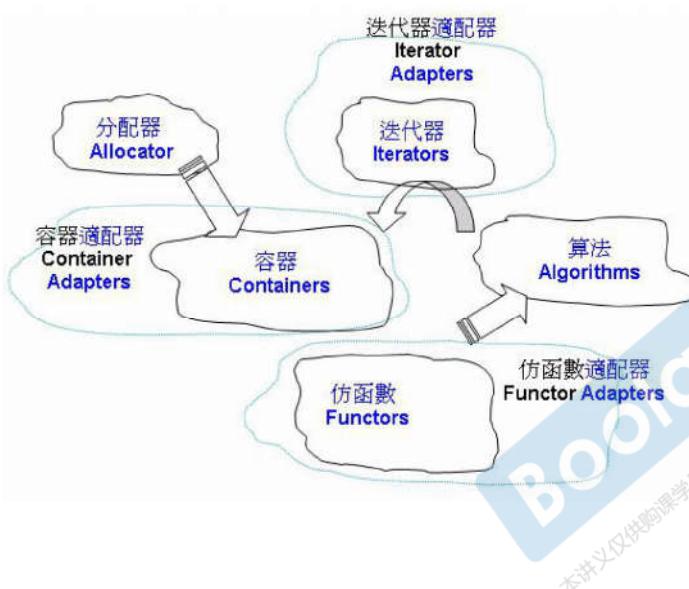
■■■■ STL 六大部件

STL 六大部件 (Components) :

- 容器 (Containers)
- 分配器 (Allocators)
- 算法 (Algorithms)
- 迭代器 (Iterators)
- 適配器 (Adapters)
- 仿函式 (Functors)



STL 六大部件關係



```
01 #include <vector>
02 #include <algorithm>
03 #include <functional>
04 #include <iostream>
05
06 using namespace std;
07
08 int main()
09 {
10     int ia[ 6 ] = { 27, 210, 12, 47, 109, 83 };
11     vector<int, allocator<int>> vi(ia,ia+6);
12
13     cout << count_if(vi.begin(), vi.end(),
14     not1(bind2nd(less<int>(), 40)));
15
16     return 0;
17 }
```

Annotations for the code:

- allocator**: Points to the `allocator<int>` part of the `vector` constructor.
- container**: Points to the `vector` type.
- iterator**: Points to the `begin()` and `end()` member functions.
- algorithm**: Points to the `count_if` function.
- function adapter (negator)**: Points to the `not1` function.
- function adapter (binder)**: Points to the `bind2nd` function.
- predicate**: Points to the `less<int>()` function object.
- function object**: Points to the `40` argument, which is a function object.

— 侯捷 —

13

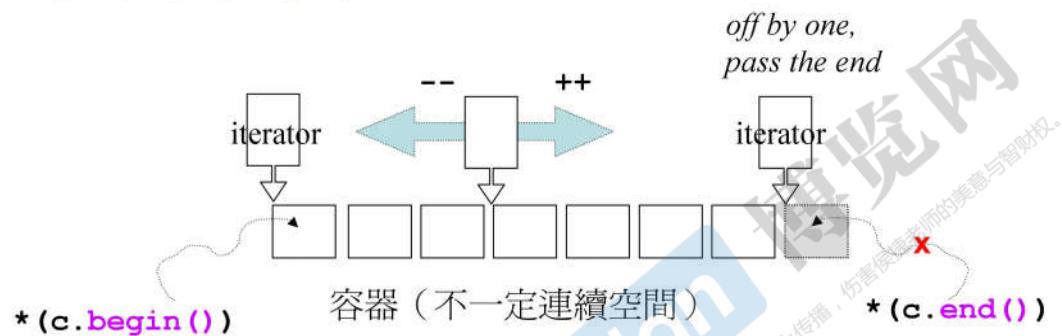
■■■ 複雜度, Complexity, Big-oh

目前常見的 Big-oh 有下列幾種情形：

1. $O(1)$ 或 $O(c)$ ：稱為常數時間(constant time)
2. $O(n)$ ：稱為線性時間(linear time)
3. $O(\log_2 n)$ ：稱為次線性時間(sub-linear time)
4. $O(n^2)$ ：稱為平方時間(quadratic time)
5. $O(n^3)$ ：稱為立方時間(cubic time)
6. $O(2^n)$ ：稱為指數時間(exponential time)
7. $O(n \log_2 n)$ ：介於線性及二次方成長的中間之行為模式。

■■■■ “前閉後開” 區間

[] () []



```
Container<T> c;  
...  
Container<T>::iterator ite = c.begin();  
for (; ite != c.end(); ++ite)  
    ...
```

|||| range-based for statement (since C++11)

```
for ( decl : coll ) {  
    statement  
}
```

```
for ( int i : { 2, 3, 5, 7, 9, 13, 17, 19 } ) {  
    std::cout << i << std::endl;  
}
```

```
std::vector<double> vec;  
...  
for ( auto elem : vec ) {  
    std::cout << elem << std::endl;  
}  
  
for ( auto& elem : vec ) {  
    elem *= 3;  
}
```

— 侯捷 —

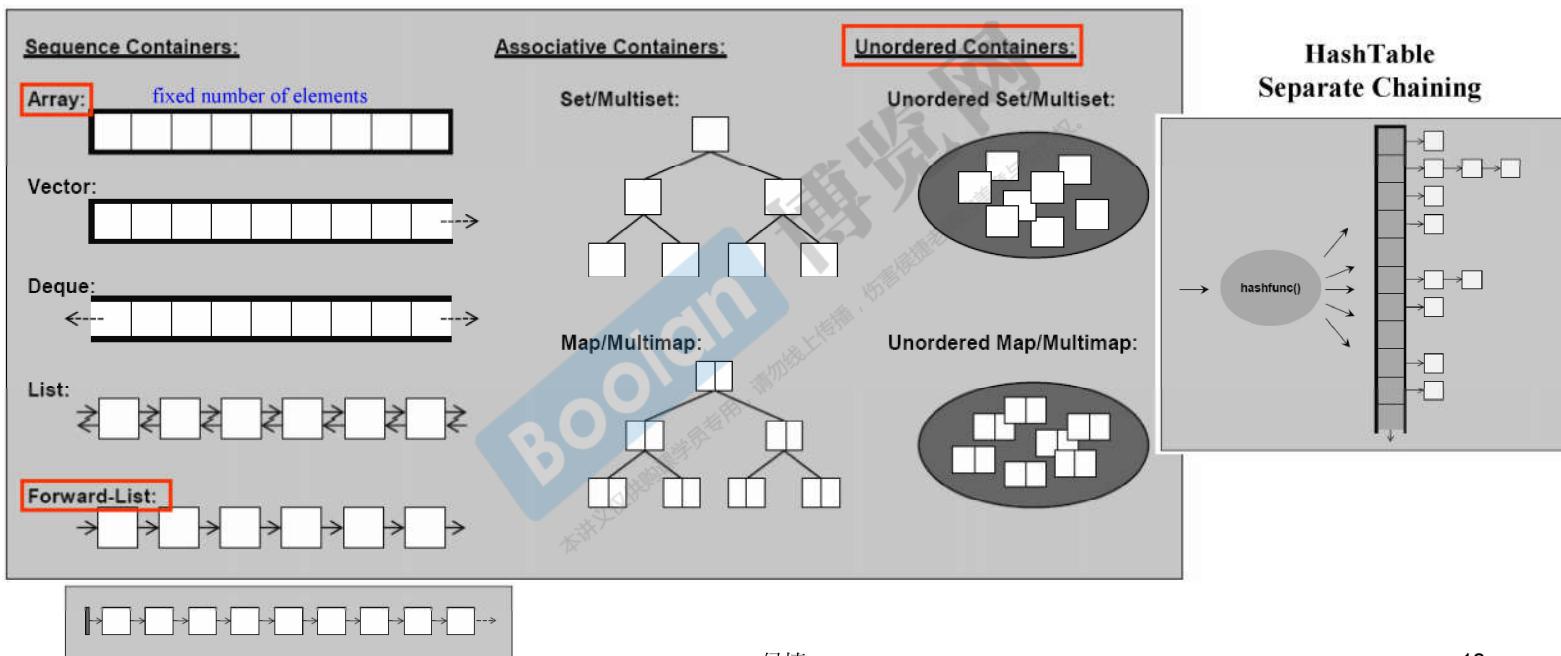
|||| auto keyword (since C++11)

```
list<string> c;  
...  
list<string>::iterator ite;  
ite = ::find(c.begin(), c.end(), target);
```



```
list<string> c;  
...  
auto ite = ::find(c.begin(), c.end(), target);
```

容器 – 結構與分類



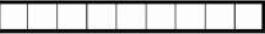
— 侯捷 —

以下測試程序之 輔助函數

```
11  using std::cin;
12  using std::cout;
13  using std::string;
14
15  long get_a_target_long()
16  {
17      long target=0;
18
19      cout << "target (0~" << RAND_MAX << "): ";
20      cin >> target;
21      return target;
22  }
23
24  string get_a_target_string()
25  {
26      long target=0;
27      char buf[10];
28
29      cout << "target (0~" << RAND_MAX << "): ";
30      cin >> target;
31      sprintf(buf, 10, "%d", target);
32      return string(buf);
33  }
```

```
35  int compareLongs(const void* a, const void* b)
36  {
37      return ( *(long*)a - *(long*)b );
38  }
39
40  int compareStrings(const void* a, const void* b)
41  {
42      if ( *(string*)a > *(string*)b )
43          return 1;
44      else if ( *(string*)a < *(string*)b )
45          return -1;
46      else
47          return 0;
48  }
```

使用容器 array

Array:


```
51 #include <array>
52 #include <iostream>
53 #include <ctime>
54 #include <cstdlib> //qsort, bsearch, NULL
55 namespace jj01
56 {
57     void test_array()
58     {
59         cout << "\ntest_array()..... \n";
60
61         array<long, ASIZE> c;
62
63         clock_t timeStart = clock();
64         for(long i=0; i< ASIZE; ++i) {
65             c[i] = rand();
66         }
67         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
68         cout << "array.size()= " << c.size() << endl;
69         cout << "array.front()= " << c.front() << endl;
70         cout << "array.back()= " << c.back() << endl;
71         cout << "array.data()= " << c.data() << endl;
72
73         long target = get_a_target_long();
74
75         timeStart = clock();
76         qsort(c.data(), ASIZE, sizeof(long), compareLongs);
77         long* pItem = (long*)bsearch(&target, (c.data()), ASIZE, sizeof(long), compareLongs);
78         cout << "qsort()+bsearch(), milli-seconds : " << (clock()-timeStart) << endl; //
79         if (pItem != NULL)
80             cout << "found, " << *pItem << endl;
81         else
82             cout << "not found! " << endl;
83     }
84 }
```

```
D:\handout\C++11-test-DevC++\Test-STL\test-stl.exe
select: 1

test_array()..... .
milli-seconds : 47
array.size()= 500000
array.front()= 3557
array.back()= 23084
array.data()= 0x47a20
target <0x32767>: 20000
qsort()>+bsearch(), milli-seconds : 187
found, 20000
```

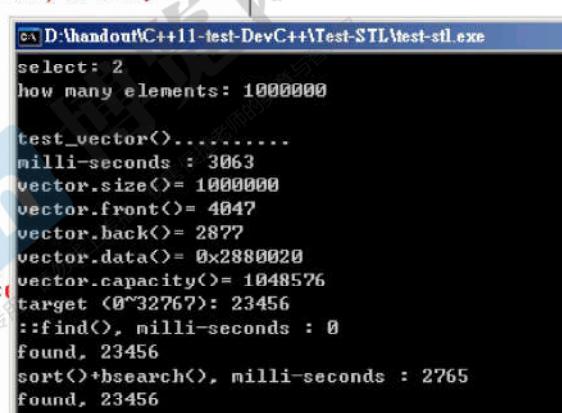
使用容器 vector

```
Vector:   
D:\handout\c++11-test-DevC++\Test-STL\test-stl.exe  
select: 2  
how many elements: 1000000  
  
test_vector().....  
milli-seconds : 3063  
vector.size()= 1000000  
vector.front()= 4047  
vector.back()= 2877  
vector.data()= 0x2880020  
vector.capacity()= 1048576  
target (0"32767): 23456  
::find(), milli-seconds : 0  
found, 23456  
sort()>bsearch(), milli-seconds : 2765  
found, 23456
```

```
86 #include <vector>  
87 #include <stdexcept>  
88 #include <string>  
89 #include <cstdlib> //abort()  
90 #include <cstdio> //snprintf()  
91 #include <iostream>  
92 #include <ctime>  
93 #include <algorithm> //sort()  
94 namespace jj02  
95 {  
96 void test_vector(long& value)  
97 {  
98 cout << "\ntest_vector()..... \n";  
99  
100 vector<string> c;  
101 char buf[10];  
102  
103 clock_t timeStart = clock();  
104 for(long i=0; i< value; ++i)  
105 {  
106 try {  
107 snprintf(buf, 10, "%d", rand());  
108 c.push_back(string(buf));  
109 }  
110 catch(exception& p) {  
111 cout << "i=" << i << " " << p.what() << endl;  
112 //曾經最高 i=58389486 then std::bad_alloc  
113 abort();  
114 }  
115 }  
116 cout << "milli-seconds : " << (clock()-timeStart) << endl;  
117 cout << "vector.size()= " << c.size() << endl;  
118 cout << "vector.front()= " << c.front() << endl;  
119 cout << "vector.back()= " << c.back() << endl;  
120 cout << "vector.data()= " << c.data() << endl;  
121 cout << "vector.capacity()= " << c.capacity() << endl;  
122  
123 }
```

使用容器 vector

```
124 string target = get_a_target_string();
125 {
126     timeStart = clock();
127     auto pItem = ::find(c.begin(), c.end(), target);
128     cout << "::find()", milli-seconds : " << (clock()-timeStart) << endl;
129
130     if (pItem != c.end())
131         cout << "found, " << *pItem << endl;
132     else
133         cout << "not found! " << endl;
134 }
135
136 {
137     timeStart = clock();
138     sort(c.begin(), c.end());
139     string* pItem = (string*)bsearch(&target, (c.data()),
140                                     c.size(), sizeof(string));
141     cout << "sort()>bsearch()", milli-seconds : " << (clock()
142
143     if (pItem != NULL)
144         cout << "found, " << *pItem << endl;
145     else
146         cout << "not found! " << endl;
147 }
148
149 }
```



D:\handout\c++11-test\DevC++\Test-STL\test-stl.exe
select: 2
how many elements: 1000000

test_vector<...>.....
milli-seconds : 3063
vector.size()= 1000000
vector.front()= 4047
vector.back()= 2877
vector.data()= 0x2880020
vector.capacity()= 1048576
target <0^32767>: 23456
::find(), milli-seconds : 0
found, 23456
sort()>bsearch(), milli-seconds : 2765
found, 23456

使用容器 list



```
ex D:\handout\C++11-test-DevC++\Test-STL\test
select: 3
how many elements: 1000000

test_list().....
milli-seconds : 3265
list.size()= 1000000
list.max_size()= 357913941
list.front()= 4710
list.back()= 16410
target <0~32767>: 23456
::find(), milli-seconds : 16
found, 23456
c.sort(), milli-seconds : 2312
```

```
159 namespace jj03
160 {
161     void test_list(long& value)
162     {
163         cout << "\ntest_list()..... \n";
164
165         list<string> c;
166         char buf[10];
167
168         clock_t timeStart = clock();
169         for(long i=0; i< value; ++i)
170         {
171             try {
172                 sprintf(buf, 10, "%d", rand());
173                 c.push_back(string(buf));
174             }
175             catch(exception& p) {
176                 cout << "i=" << i << " " << p.what() << endl; // abort();
177             }
178         }
179         cout << "milli-seconds : " << (clock()-timeStart) << endl;
180         cout << "list.size()= " << c.size() << endl;
181         cout << "list.max_size()= " << c.max_size() << endl;
182         cout << "list.front()= " << c.front() << endl;
183         cout << "list.back()= " << c.back() << endl;
184
185         string target = get_a_target_string();
186         timeStart = clock();
187         auto pItem = ::find(c.begin(), c.end(), target);
188         cout << "::find(), milli-seconds : " << (clock()-timeStart) << endl;
189
190         if (pItem != c.end())
191             cout << "found, " << *pItem << endl;
192         else
193             cout << "not found! " << endl;
194
195         timeStart = clock();
196         c.sort();
197         cout << "c.sort(), milli-seconds : " << (clock()-timeStart) << endl;
198
199     } //又一陣子才離開函數; 它在 destroy...
200 }
```

/* Returns the number of elements in the %list. */
size_type
size() const _GLIBCXX_NOEXCEPT
{ return std::distance(begin(), end()); }

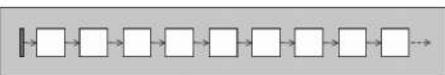
/* Returns the size() of the largest possible %list.
size_type
max_size() const _GLIBCXX_NOEXCEPT
{ return _M_get_Node_allocator().max_size(); }

size_type
max_size() const _GLIBCXX_USE_NOEXCEPT
{ return size_t(-1) / sizeof(_Tp); }

std::allocator<_Tp>, 亦即
new_allocator, 內有:



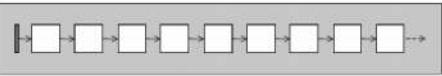
使用容器 `forward_list`



```
c:\>D:\handout\c++11-test-DevC++\Test-STL\test-  
select: 4  
how many elements: 1000000  
  
test_forward_list<>.....  
milli-seconds : 3204  
forward_list.max_size()= 536870911  
forward_list.front()= 8180  
target <0~32767>: 23456  
::find(), milli-seconds : 15  
found, 23456  
c.sort(), milli-seconds : 2656
```

```
210 namespace jj04
211 {
212     void test_forward_list(long& value)
213     {
214         cout << "\ntest_forward_list()..... \n";
215
216         forward_list<string> c;
217         char buf[10];
218
219         clock_t timeStart = clock();
220         for(long i=0; i< value; ++i)
221         {
222             try {
223                 sprintf(buf, 10, "%d", rand());
224                 c.push_front(string(buf));
225             }
226             catch(exception& p) {
227                 cout << "i=" << i << " " << p.what() << endl; // abort();
228             }
229         }
230         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
231         cout << "forward_list.max_size()=" << c.max_size() << endl;
232         cout << "forward_list.front()=" << c.front() << endl;
233         // cout << "forward_list.back()=" << c.back() << endl; //no such member function
234         // cout << "forward_list.size()=" << c.size() << endl; //no such member function
235
236
237
238         string target = get_a_target_string();
239         timestamp = clock();
240         auto pItem = ::find(c.begin(), c.end(), target);
241         cout << "::find(), milli-seconds : " << (clock()-timeStart) << endl; //
242
243         if (pItem != c.end())
244             cout << "found, " << *pItem << endl;
245         else
246             cout << "not found! " << endl;
247
248         timeStart = clock();
249         c.sort();
250         cout << "c.sort(), milli-seconds : " << (clock()-timeStart) << endl; //
251     }
252 }
```

使用容器 slist

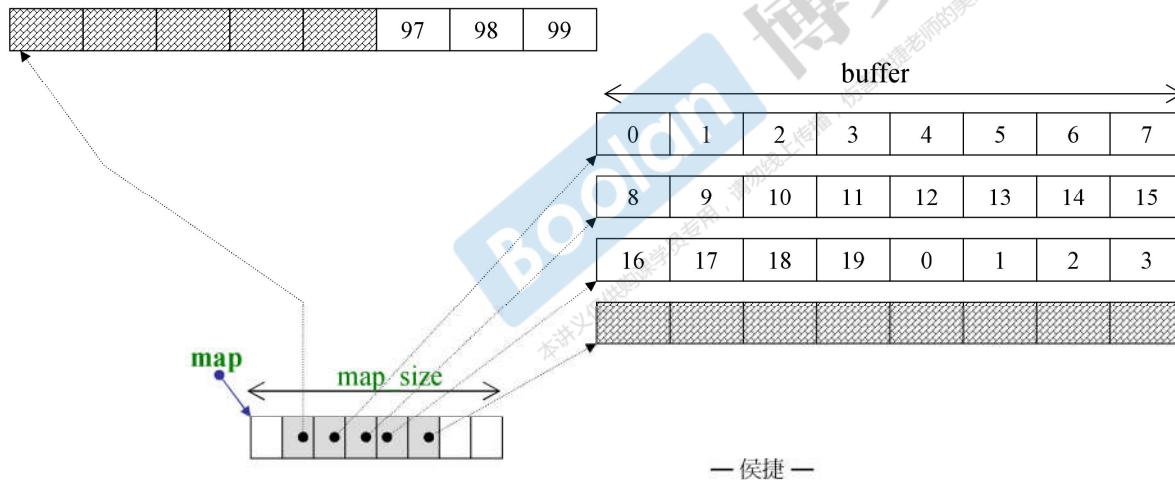
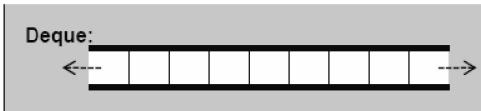


```
c:\D:\handout\c++11-test-DevC++\Test-S>
select: 10
how many elements: 1000000
test_slist()..... .
milli-seconds : 3046
```

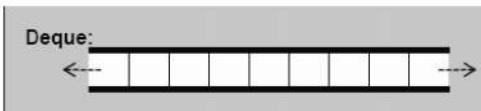
```
508 #include <ext\list>
509 #include <stdexcept>
510 #include <string>
511 #include <cstdlib> //abort()
512 #include <cstdio> //snprintf()
513 #include <iostream>
514 #include <ctime>
515 namespace jj10
516 {
517 void test_slist(long& value)
518 {
519     cout << "\ntest_slist()..... \n";
520
521     __gnu_cxx::slist<string> c;
522     char buf[10];
523
524     clock_t timeStart = clock();
525     for(long i=0; i< value; ++i)
526     {
527         try
528         {
529             sprintf(buf, 10, "%d", rand());
530             c.push_front(string(buf));
531         }
532         catch(exception& p)
533         {
534             cout << "i=" << i << " " << p.what() << endl; // abort();
535         }
536     }
537     cout << "milli-seconds : " << (clock()-timeStart) << endl;
538 }
539 }
```



使用容器 deque



使用容器 deque

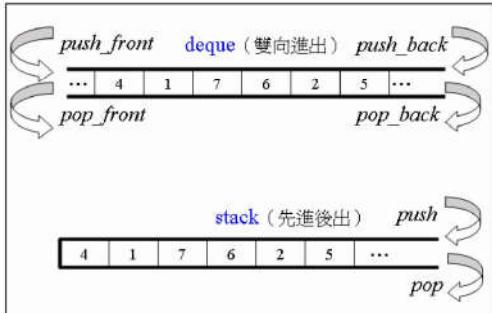


```
D:\handout\c++11-test\DevC++\Test-STL\test>
select: 5
how many elements: 1000000

test_deque().....
milli-seconds : 2704
deque.size()= 1000000
deque.front()= 5249
deque.back()= 1098
deque.max_size()= 1073741823
target <0^32767>: 23456
::find(), milli-seconds : 15
found, 23456
::sort(), milli-seconds : 3110
```

```
261 namespace jj05
262 {
263     void test_deque(long& value)
264     {
265         cout << "\ntest_deque()..... \n";
266
267         deque<string> c;
268         char buf[10];
269
270         clock_t timeStart = clock();
271         for(long i=0; i< value; ++i)
272         {
273             try {
274                 sprintf(buf, 10, "%d", rand());
275                 c.push_back(string(buf));
276             }
277             catch(exception& p) {
278                 cout << "i=" << i << " " << p.what() << endl; // abort();
279             }
280         }
281         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
282         cout << "deque.size()= " << c.size() << endl;
283         cout << "deque.front()= " << c.front() << endl;
284         cout << "deque.back()= " << c.back() << endl;
285         cout << "deque.max_size()= " << c.max_size() << endl;
286
287         string target = get_a_target_string();
288         timeStart = clock();
289         auto pItem = ::find(c.begin(), c.end(), target);
290         cout << "::find(), milli-seconds : " << (clock()-timeStart) << endl;
291
292         if (pItem != c.end())
293             cout << "found: " << *pItem << endl;
294         else
295             cout << "not found! " << endl;
296
297         timeStart = clock();
298         ::sort(c.begin(), c.end());
299         cout << "::sort(), milli-seconds : " << (clock()-timeStart) << endl;
300
301     }
302 }
```

使用容器 stack



```
c:\D\handout\C++11-test>DevC++\Test  
select: 17  
how many elements: 300000  
  
test_stack().....  
milli-seconds : 812  
stack.size()= 300000  
stack.top()= 23929  
stack.size()= 299999  
stack.top()= 12911
```

```
847 namespace jj17  
848 {  
849     void test_stack(long& value)  
850     {  
851         cout << "\ntest_stack()..... \n";  
852         stack<string> c;  
853         char buf[10];  
854  
855         clock_t timeStart = clock();  
856         for(long i=0; i< value; ++i)  
857         {  
858             try {  
859                 snprintf(buf, 10, "%d", rand());  
860                 c.push(string(buf));  
861             }  
862             catch(exception& p){  
863                 cout << "i=" << i << " " << p.what() << endl;  
864                 abort();  
865             }  
866         }  
867         cout << "milli-seconds : " << (clock()-timeStart) << endl;  
868         cout << "stack.size()= " << c.size() << endl;  
869         cout << "stack.top()= " << c.top() << endl;  
870         c.pop();  
871         cout << "stack.size()= " << c.size() << endl;  
872         cout << "stack.top()= " << c.top() << endl;  
873     }  
874 }  
875 }
```

使用容器 queue

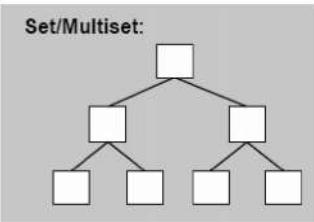


```
ex D:\handoutC++11-test-DevC++\Test-
select: 18
how many elements: 300000

test_queue().....
milli-seconds : 890
queue.size()= 300000
queue.front()= 12074
queue.back()= 14585
queue.size()= 299999
queue.front()= 6888
queue.back()= 14585
```

```
884 namespace jj18
885 {
886     void test_queue(long& value)
887     {
888         cout << "\ntest_queue()..... \n";
889
890         queue<string> c;
891         char buf[10];
892
893         clock_t timeStart = clock();
894         for(long i=0; i< value; ++i)
895         {
896             try {
897                 snprintf(buf, 10, "%d", rand());
898                 c.push(string(buf));
899             }
900             catch(exception& p) {
901                 cout << "i=" << i << " " << p.what() << endl;
902                 abort();
903             }
904         }
905         cout << "milli-seconds : " << (clock()-timeStart) << endl;
906         cout << "queue.size()= " << c.size() << endl;
907         cout << "queue.front()= " << c.front() << endl;
908         cout << "queue.back()= " << c.back() << endl;
909         c.pop();
910         cout << "queue.size()= " << c.size() << endl;
911         cout << "queue.front()= " << c.front() << endl;
912         cout << "queue.back()= " << c.back() << endl;
913     }
914 }
```

使用容器 multiset

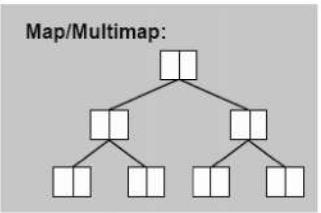


```
ex D:\handout\c++11-test-DevC++\Test-STL>
select: 6
how many elements: 1000000

test_multiset<...>.....
milli-seconds : 6609
multiset.size()= 1000000
multiset.max_size()= 214748364
target <0~32767>: 23456
::find(), milli-seconds : 203
found, 23456
c.find(), milli-seconds : 0
found, 23456
```

```
313 void test_multiset(long& value)
314 {
315     cout << "\ntest_multiset()..... \n";
316
317     multiset<string> c;
318     char buf[10];
319     clock_t timeStart = clock();
320     for(long i=0; i< value; ++i)
321     {
322         try {
323             sprintf(buf, 10, "%d", rand());
324             c.insert(string(buf));
325         }
326         catch(exception& p) {
327             cout << "i=" << i << " " << p.what() << endl; // abort();
328         }
329     }
330     cout << "milli-seconds : " << (clock()-timeStart) << endl; //
331     cout << "multiset.size()= " << c.size() << endl;
332     cout << "multiset.max_size()= " << c.max_size() << endl;
333
334     string target = get_a_target_string();
335     {
336         timeStart = clock();
337         auto pItem = ::find(c.begin(), c.end(), target); //比 c.find(...) 慢很多
338         cout << "::find(), milli-seconds : " << (clock()-timeStart) << endl;
339         if (pItem != c.end())
340             cout << "found, " << *pItem << endl;
341         else
342             cout << "not found! " << endl;
343     }
344     {
345         timeStart = clock(); //比 ::find(...) 快很多
346         auto pItem = c.find(target);
347         cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;
348         if (pItem != c.end())
349             cout << "found, " << *pItem << endl;
350         else
351             cout << "not found! " << endl;
352     }
353 }
```

使用容器 multimap



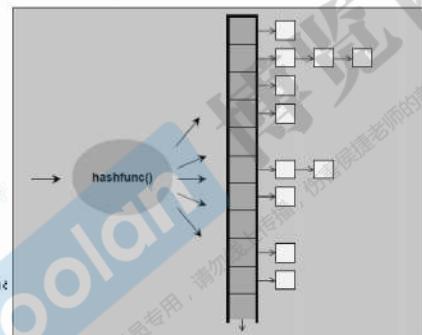
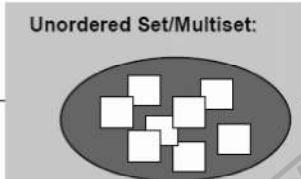
```
ex D:\handout\C++11-test-DevC++\Test-STL>
select: 7
how many elements: 1000000

test_multimap<>. .....
milli-seconds : 4812
multimap.size()= 1000000
multimap.max_size()= 178956970
target <0~32767>: 23456
c.find(), milli-seconds : 0
found, value=29247
```

```
364 namespace jj07
365 {
366     void test_multimap(long& value)
367     {
368         cout << "\ntest_multimap()..... \n";
369
370         multimap<long, string> c;
371         char buf[10];
372
373         clock_t timeStart = clock();
374         for(long i=0; i< value; ++i)
375         {
376             try {
377                 snprintf(buf, 10, "%d", rand());
378                 //multimap 不可使用 [] 做 insertion
379                 c.insert(pair<long, string>(i,buf));
380             }
381             catch(exception& p) {
382                 cout << "i=" << i << " " << p.what() << endl; //
383                 abort();
384             }
385         }
386         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
387         cout << "multimap.size()= " << c.size() << endl;
388         cout << "multimap.max_size()= " << c.max_size() << endl;
389
390         long target = get_a_target_long();
391         timeStart = clock();
392         auto pItem = c.find(target);
393         cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;
394         if (pItem != c.end())
395             cout << "found, value=" << (*pItem).second << endl;
396         else
397             cout << "not found! " << endl;
398     }
399 }
```

使用容器 unordered_multiset

```
408 namespace jj08
409 {
410     void test_unordered_multiset(long& value)
411     {
412         cout << "\ntest_unordered_multiset()..... \n";
413
414         unordered_multiset<string> c;
415         char buf[10];
416
417         clock_t timeStart = clock();
418         for(long i=0; i< value; ++i)
419         {
420             try {
421                 snprintf(buf, 10, "%d", rand());
422                 c.insert(string(buf));
423             }
424             catch(exception& p) {
425                 cout << "i=" << i << " " << p.what();
426                 abort();
427             }
428         }
429         cout << "milli-seconds : " << (clock()-timeStart) << endl; // 
430         cout << "unordered_multiset.size()= " << c.size() << endl;
431         cout << "unordered_multiset.max_size()= " << c.max_size() << endl;
432         cout << "unordered_multiset.bucket_count()= " << c.bucket_count() << endl;
433         cout << "unordered_multiset.load_factor()= " << c.load_factor() << endl;
434         cout << "unordered_multiset.max_load_factor()= " << c.max_load_factor() << endl;
435         cout << "unordered_multiset.max_bucket_count()= " << c.max_bucket_count() << endl;
436         for (unsigned i=0; i< 20; ++i) {
437             cout << "bucket #" << i << " has " << c.bucket_size(i) << " elements.\n";
438     }
```



```
Unordered Set/Multiset:
select: 8
how many elements: 1000000
test_unordered_multiset().....
milli-seconds : 4406
unordered_multiset.size()= 1000000
unordered_multiset.max_size()= 357913941
unordered_multiset.bucket_count()= 1056323
unordered_multiset.load_factor()= 0.94668
unordered_multiset.max_load_factor()= 1
unordered_multiset.max_bucket_count()= 357913941
bucket #0 has 0 elements.
bucket #1 has 0 elements.
bucket #2 has 0 elements.
bucket #3 has 0 elements.
bucket #4 has 0 elements.
bucket #5 has 0 elements.
bucket #6 has 0 elements.
bucket #7 has 0 elements.
bucket #8 has 0 elements.
bucket #9 has 0 elements.
bucket #10 has 0 elements.
bucket #11 has 0 elements.
bucket #12 has 24 elements.
bucket #13 has 0 elements.
bucket #14 has 0 elements.
bucket #15 has 0 elements.
bucket #16 has 0 elements.
bucket #17 has 0 elements.
bucket #18 has 0 elements.
bucket #19 has 0 elements.
target <0x32767>: 23456
::find(), milli-seconds : 109
found, 23456
c.find(), milli-seconds : 0
found, 23456
```

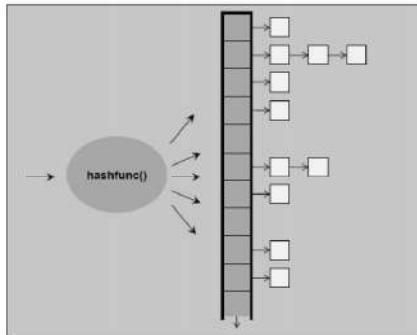
使用容器 `unordered_multiset`

```
439 string target = get_a_target_string();
440 {
441     timeStart = clock();
442     auto pItem = ::find(c.begin(), c.end(), target); //比 c.find(...) 快很多
443     cout << "::find()", milli-seconds : " << (clock()-timeStart) << endl;
444     if (pItem != c.end())
445         cout << "found, " << *pItem << endl;
446     else
447         cout << "not found! " << endl;
448 }
449 {
450     timeStart = clock();
451     auto pItem = c.find(target); //比 ::find(...) 快很多
452     cout << "c.find()", milli-seconds : " << (clock()-timeStart) << endl;
453     if (pItem != c.end())
454         cout << "found, " << *pItem << endl;
455     else
456         cout << "not found! " << endl;
457 }
458 }
459 }
460 }
```

```
c:\D:\handout\c++11-test-DevC++\Test-STL\test-stl.exe
select: 8
how many elements: 1000000

test_unordered_multiset().....
milli-seconds : 4406
unordered_multiset.size()= 1000000
unordered_multiset.max_size()= 357913941
unordered_multiset.bucket_count()= 1056323
unordered_multiset.load_factor()= 0.94668
unordered_multiset.max_load_factor()= 1
unordered_multiset.max_bucket_count()= 357913941
bucket #0 has 0 elements.
bucket #1 has 0 elements.
bucket #2 has 0 elements.
bucket #3 has 0 elements.
bucket #4 has 0 elements.
bucket #5 has 0 elements.
bucket #6 has 0 elements.
bucket #7 has 0 elements.
bucket #8 has 0 elements.
bucket #9 has 0 elements.
bucket #10 has 0 elements.
bucket #11 has 0 elements.
bucket #12 has 24 elements.
bucket #13 has 0 elements.
bucket #14 has 0 elements.
bucket #15 has 0 elements.
bucket #16 has 0 elements.
bucket #17 has 0 elements.
bucket #18 has 0 elements.
bucket #19 has 0 elements.
target <0'32767>: 23456
::find(), milli-seconds : 109
found, 23456
c.find(), milli-seconds : 0
found, 23456
```

使用容器 `unordered_multimap`

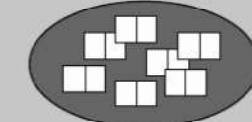


```
ex>D:\handout\c++11-test\DevC++\Test-STL\test-stl.exe
select: 9
how many elements: 1000000

test_unordered_multimap().....
milli-seconds : 4313
unordered_multimap.size()= 1000000
unordered_multimap.max_size()= 357913941
target <0^32767>: 23456
c.find(), milli-seconds : 0
found, value=15962
```

```
470 namespace jj09
471 {
472     void test_unordered_multimap(long& value)
473     {
474         cout << "\ntest_unordered_multimap()..... \n";
475
476         unordered_multimap<long, string> c;
477         char buf[10];
478
479         clock_t timeStart = clock();
480         for(long i=0; i< value; ++i)
481         {
482             try {
483                 sprintf(buf, 10, "%d", rand());
484                 //multimap 不可用 [] 进行 insertion
485                 c.insert(pair<long,string>(i,buf));
486             }
487             catch(exception& p) {
488                 cout << "i=" << i << " " << p.what() << endl; // abort();
489             }
490         }
491         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
492         cout << "unordered_multimap.size()= " << c.size() << endl;
493         cout << "unordered_multimap.max_size()= " << c.max_size() << endl;
494
495
496         long target = get_a_target_long();
497         timeStart = clock();
498         auto pItem = c.find(target);
499         cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;
500         if (pItem != c.end())
501             cout << "found, value=" << (*pItem).second << endl;
502         else
503             cout << "not found! " << endl;
504     }
505 }
506 }
```

Unordered Map/Multimap:

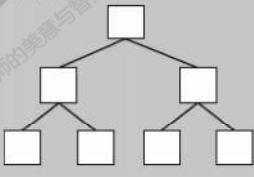


使用容器 set

```
641 namespace jj13
642 {
643     void test_set(long& value)
644     {
645         cout << "\ntest_set()..... \n";
646         set<string> c;
647         char buf[10];
648
649         clock_t timeStart = clock();
650         for(long i=0; i< value; ++i)
651         {
652             try {
653                 snprintf(buf, 10, "%d", rand());
654                 c.insert(string(buf));
655             }
656             catch(exception& p) {
657                 cout << "i=" << i << " " << p.what() << endl;
658                 abort();
659             }
660         }
661         cout << "milli-seconds : " << (clock()-timeStart) << endl;
662         cout << "set.size()= " << c.size() << endl;
663         cout << "set.max_size()= " << c.max_size() << endl;
664
665         string target = get_a_target_string();
666         {
667             timeStart = clock();
668             auto pItem = ::find(c.begin(), c.end(), target); //比 c.find(...) 慢很多
669             cout << "::find(), milli-seconds : " << (clock()-timeStart) << endl;
670             if (pItem != c.end())
671                 cout << "found, " << *pItem << endl;
672             else
673                 cout << "not found! " << endl;
674         }
675     }
}
```

```
676 {
677     timeStart = clock();
678     auto pItem = c.find(target); //比 ::find(...) 快很多
679     cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;
680     if (pItem != c.end())
681         cout << "found, " << *pItem << endl;
682     else
683         cout << "not found! " << endl;
684 }
685 }
686 }
687 }
```

Set/Multiset:

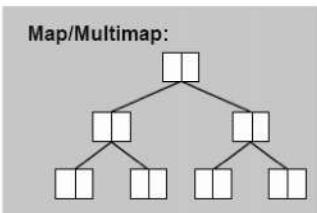


```
D:\handout\c++11-test-DevC++\Test-STL
select: 13
how many elements: 1000000

test_set().....
milli-seconds : 3922
set.size()= 32768
set.max_size()= 214748364
target (<0~32767): 23456
::find(), milli-seconds : 0
found, 23456
c.find(), milli-seconds : 0
found, 23456
```



使用容器 map



```
ex D:\handout\c++11-test-DevC++\Test-STI
select: 14
how many elements: 1000000

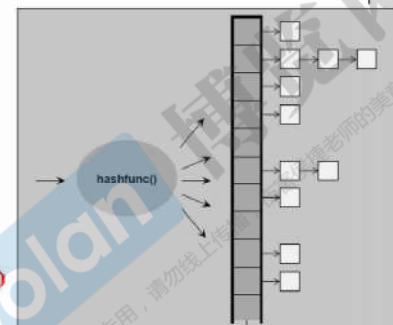
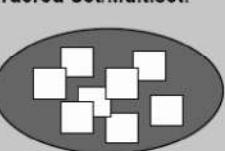
test_map().....
milli-seconds : 4890
map.size()= 1000000
map.max_size()= 178956970
target <0~32767>: 23456
c.find(), milli-seconds : 0
found, value=19128
```

```
696 namespace jj14
697 {
698     void test_map(long& value)
699     {
700         cout << "\ntest_map()..... \n";
701
702         map<long, string> c;
703         char buf[10];
704
705         clock_t timeStart = clock();
706         for(long i=0; i< value; ++i)
707         {
708             try
709             {
710                 sprintf(buf, 10, "%d", rand());
711                 c[i] = string(buf);
712             }
713             catch(exception& p)
714             {
715                 cout << "i=" << i << " " << p.what() << endl;    //
716                 abort();
717             }
718         }
719         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
720         cout << "map.size()= " << c.size() << endl;
721         cout << "map.max_size()= " << c.max_size() << endl;
722
723         long target = get_a_target_long();
724         timeStart = clock();
725         auto pItem = c.find(target);
726         cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;
727         if (pItem != c.end())
728             cout << "found, value=" << (*pItem).second << endl;
729         else
730             cout << "not found! " << endl;
731     }
732 }
```

使用容器 `unordered_set`

```
739 namespace jj15
740 {
741     void test_unordered_set(long& value)
742     {
743         cout << "\ntest_unordered_set().....\n";
744
745         unordered_set<string> c;
746         char buf[10];
747
748         clock_t timeStart = clock();
749         for(long i=0; i< value; ++i)
750         {
751             try {
752                 sprintf(buf, 10, "%d", rand());
753                 c.insert(string(buf));
754             }
755             catch(exception& p) {
756                 cout << "i=" << i << " " << p.what();
757                 abort();
758             }
759         }
760         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
761         cout << "unordered_set.size()= " << c.size() << endl;
762         cout << "unordered_set.max_size()= " << c.max_size() << endl;
763         cout << "unordered_set.bucket_count()= " << c.bucket_count() << endl;
764         cout << "unordered_set.load_factor()= " << c.load_factor() << endl;
765         cout << "unordered_set.max_load_factor()= " << c.max_load_factor() << endl;
766         cout << "unordered_set.max_bucket_count()= " << c.max_bucket_count() << endl;
767         for (unsigned i=0; i< 20; ++i) {
768             cout << "bucket #" << i << " has " << c.bucket_size(i) << " elements.\n";
769     }
```

Unordered Set/Multiset:



```
D:\handout\C++11-test\DevC++\VTest-STL\test-stl.exe
select: 15
how many elements: 1000000

test_unordered_set().....
milli-seconds : 2891
unordered_set.size()= 32768
unordered_set.max_size()= 357913941
unordered_set.bucket_count()= 62233
unordered_set.load_factor()= 0.526537
unordered_set.max_load_factor()= 1
unordered_set.max_bucket_count()= 357913941
bucket #0 has 1 elements.
bucket #1 has 1 elements.
bucket #2 has 1 elements.
bucket #3 has 0 elements.
bucket #4 has 3 elements.
bucket #5 has 0 elements.
bucket #6 has 0 elements.
bucket #7 has 1 elements.
bucket #8 has 0 elements.
bucket #9 has 0 elements.
bucket #10 has 0 elements.
bucket #11 has 1 elements.
bucket #12 has 0 elements.
bucket #13 has 2 elements.
bucket #14 has 1 elements.
bucket #15 has 1 elements.
bucket #16 has 1 elements.
bucket #17 has 1 elements.
bucket #18 has 0 elements.
bucket #19 has 0 elements.
target <0^32767>: 23456
::find(), milli-seconds : 0
found, 23456
c.find(), milli-seconds : 0
found, 23456
```

使用容器 unordered_set

```
770 string target = get_a_target_string();
771 {
772     timeStart = clock();
773     auto pItem = ::find(c.begin(), c.end(), target); //比 c.find(...) 慢很多
774     cout << "::find(), milli-seconds : " << (clock() - timeStart) << endl;
775     if (pItem != c.end())
776         cout << "found, " << *pItem << endl;
777     else
778         cout << "not found! " << endl;
779 }
780
781 {
782     timeStart = clock();
783     auto pItem = c.find(target); //比 ::find(...) 快很多
784     cout << "c.find(), milli-seconds : " << (clock() - timeStart) << endl;
785     if (pItem != c.end())
786         cout << "found, " << *pItem << endl;
787     else
788         cout << "not found! " << endl;
789 }
790
791 }
```

```
D:\handout\C++11-test-DevC++\Test-STL\test-stl.exe
select: 15
how many elements: 1000000

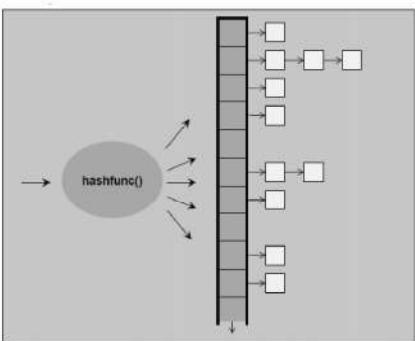
test_unordered_set().....
milli-seconds : 2891
unordered_set.size()= 32768
unordered_set.max_size()= 357913941
unordered_set.bucket_count()= 62233
unordered_set.load_factor()= 0.526537
unordered_set.max_load_factor()= 1
unordered_set.max_bucket_count()= 357913941
bucket #0 has 1 elements.
bucket #1 has 1 elements.
bucket #2 has 1 elements.
bucket #3 has 0 elements.
bucket #4 has 3 elements.
bucket #5 has 0 elements.
bucket #6 has 0 elements.
bucket #7 has 1 elements.
bucket #8 has 0 elements.
bucket #9 has 0 elements.
bucket #10 has 0 elements.
bucket #11 has 1 elements.
bucket #12 has 0 elements.
bucket #13 has 2 elements.
bucket #14 has 1 elements.
bucket #15 has 1 elements.
bucket #16 has 1 elements.
bucket #17 has 1 elements.
bucket #18 has 0 elements.
bucket #19 has 0 elements.
target <0x32767>: 23456
::find(), milli-seconds : 0
found, 23456
c.find(), milli-seconds : 0
found, 23456
```

— 侯捷 —

38



使用容器 unordered_map

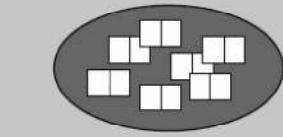


```
D:\handout\c++11-test\DevC++\Test-STL\test-stl.c
select: 16
how many elements: 1000000

test_unordered_map().....
milli-seconds : 3797
unordered_map.size()= 1000000
unordered_map.max_size()= 357913941
target <0~32767>: 23456
c.find(), milli-seconds : 0
found, value=12218
```

```
801 namespace jj16
802 {
803     void test_unordered_map(long& value)
804     {
805         cout << "\ntest_unordered_map()..... \n";
806
807         unordered_map<long, string> c;
808         char buf[10];
809
810         clock_t timeStart = clock();
811         for(long i=0; i< value; ++i)
812         {
813             try {
814                 sprintf(buf, 10, "%d", rand());
815                 c[i] = string(buf);
816             }
817             catch(exception& p) {
818                 cout << "i=" << i << " " << p.what() << endl;
819                 abort();
820             }
821         }
822         cout << "milli-seconds : " << (clock()-timeStart) << endl;
823         cout << "unordered_map.size()= " << c.size() << endl;
824         cout << "unordered_map.max_size()= " << c.max_size() << endl;
825
826
827         long target = get_a_target_long();
828         timeStart = clock();
829         //auto pItem = find(c.begin(), c.end(), target); //map 不適用 ::find()
830         auto pItem = c.find(target);
831
832         cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;
833         if (pItem != c.end())
834             cout << "found, value=" << (*pItem).second << endl;
835         else
836             cout << "not found! " << endl;
837     }
838 }
```

Unordered Map/Multimap:





使用容器 **hash_set**

hash_map

hash_multiset

hash_multimap





使用分配器 allocator

```
template<typename _Tp, typename _Alloc = std::allocator<_Tp>>
class vector : protected _Vector_base<_Tp, _Alloc>
```

```
template<typename _Tp, typename _Alloc = std::allocator<_Tp>>
class list : protected _List_base<_Tp, _Alloc>
```

```
template<typename _Tp, typename _Alloc = std::allocator<_Tp>>
class deque : protected _Deque_base<_Tp, _Alloc>
```

```
template<typename _Key, typename _Compare = std::less<_Key>,
         typename _Alloc = std::allocator<_Key>>
class set
```

```
template <typename _Key, typename _Tp, typename _Compare = std::less<_Key>,
          typename _Alloc = std::allocator<std::pair<const _Key, _Tp>>>
class map
```

```
template<class _Value,
         class _Hash = hash<_Value>,
         class _Pred = std::equal_to<_Value>,
         class _Alloc = std::allocator<_Value>>
class unordered_set
```

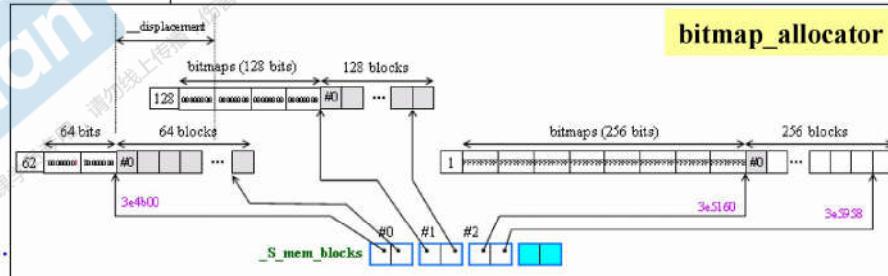
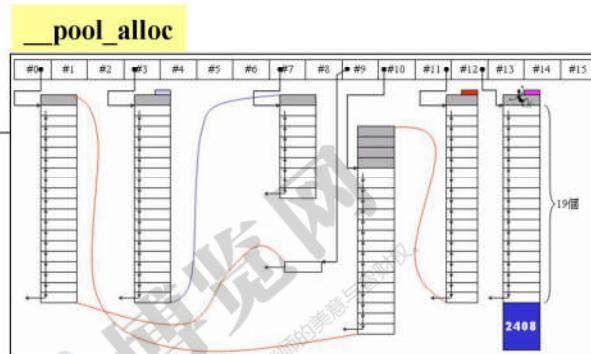
```
template<class _Key, class _Tp,
         class _Hash = hash<_Key>,
         class _Pred = std::equal_to<_Key>,
         class _Alloc = std::allocator<std::pair<const _Key, _Tp>>>
class unordered_map
```

使用分配器 allocator

```

916 #include <list>
917 #include <stdexcept>
918 #include <string>
919 #include <cstdlib>           //abort()
920 #include <cstdio>            //snprintf()
921 #include <algorithm>         //find()
922 #include <iostream>
923 #include <ctime>
924
925 #include <cstddef>
926 #include <memory>           //內含 std::allocator
927 //欲使用 std::allocator 以外的 allocator, 得自行 #include <ext>...
928 #include <ext\array_allocator.h>
929 #include <ext\mt_allocator.h>
930 #include <ext\debug_allocator.h>
931 #include <ext\pool_allocator.h>
932 #include <ext\bitmap_allocator.h>
933 #include <ext\malloc_allocator.h>
934 #include <ext\new_allocator.h>
935 namespace jj20
936 {
937     void test_list_with_special_allocator()
938     {
939         cout << "\ntest_list_with_special_allocator()....."
940
941         list<string, allocator<string>> c1;
942         list<string, __gnu_cxx::__malloc_allocator<string>> c2;
943         list<string, __gnu_cxx::__new_allocator<string>> c3;
944         list<string, __gnu_cxx::__pool_alloc<string>> c4;
945         list<string, __gnu_cxx::__mt_alloc<string>> c5;
946         list<string, __gnu_cxx::__bitmap_allocator<string>> c6;

```



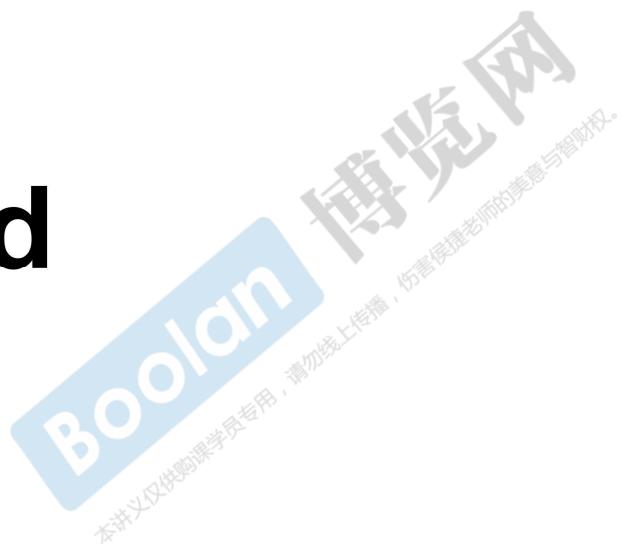
使用分配器 allocator

```
948 int choice;
949 long value;
950
951     cout << "select: ";
952     cin >> choice;
953     if ( choice != 0 ) {
954         cout << "how many elements: ";
955         cin >> value;
956     }
957
958     char buf[10];
959     clock_t timeStart = clock();
960     for(long i=0; i< value; ++i)
961     {
962         try {
963             snprintf(buf, 10, "%d", i);
964             switch (choice)
965             {
966                 case 1 : c1.push_back(string(buf));
967                             break;
968                 case 2 : c2.push_back(string(buf));
969                             break;
970                 case 3 : c3.push_back(string(buf));
971                             break;
972                 case 4 : c4.push_back(string(buf));
973                             break;
974                 case 5 : c5.push_back(string(buf));
975                             break;
976                 case 6 : c6.push_back(string(buf));
977                             break;
978                 default:
979                     break;
980             }
981         } catch(exception& p) {
982             cout << "i=" << i << " " << p.what() << endl;
983             abort();
984         }
985     }
986
987     cout << "a lot of push_back(), milli-seconds : "
988         << (clock()-timeStart) << endl;
```

```
989
990 //test all allocators' allocate() & deallocate();
991 int* p;
992 allocator<int> alloc1;
993 p = alloc1.allocate(1);
994 alloc1.deallocate(p,1);
995
996 __gnu_cxx::malloc_allocator<int> alloc2;
997 p = alloc2.allocate(1);
998 alloc2.deallocate(p,1);
999
1000 __gnu_cxx::new_allocator<int> alloc3;
1001 p = alloc3.allocate(1);
1002 alloc3.deallocate(p,1);
1003
1004 __gnu_cxx::__pool_alloc<int> alloc4;
1005 p = alloc4.allocate(2);
1006 alloc4.deallocate(p,2);
1007
1008 __gnu_cxx::__mt_alloc<int> alloc5;
1009 p = alloc5.allocate(1);
1010 alloc5.deallocate(p,1);
1011
1012 __gnu_cxx::bitmap_allocator<int> alloc6;
1013 p = alloc6.allocate(3);
1014 alloc6.deallocate(p,3); |
1015
1016 }
```

```
1017 }
```

The End



— 侯捷 —

44

C++ 標準庫

體系結構與內核分析

(C++ Standard Library — architecture & sources)

第二講



侯捷

— 侯捷 —

45

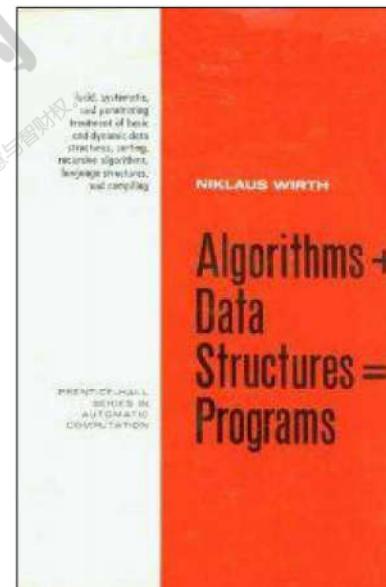
源碼之前
了無秘密



■■■■ 你應具備的基礎

- C++ 基本語法
- 模板 (Templates) 基礎
令你事半功倍
- 數據結構 (Data Structures) 和算法 (Algorithms) 概念，
令你如魚得水

1976 book
written by
Niklaus Wirth



■■■■ 標準庫版本, Visual C++

(Visual C++ 6.0 畫面)

BoostSmartPtr - Microsoft Visual C++ - [E:\...\BoostSmartPtrTest.cpp]

File Edit View Insert Project Build Tools Window Help

pimpl_sample::imp | All class members | func

```
// <Beyond the C++ Standard Library, An Introduction to Boost>, Chap1
#include <iostream>
#include "boost/scoped_ptr.hpp"
#include "boost\shared_ptr.hpp"

using namespace std;
using namespace boost;

//p12-
class pimpl_sample {
private:
    struct impl {
        void func() { cout << s << endl; }
        string s;
    };
    impl* pimpl;
public:
};

-----Configuration: BoostSmartPtr - Win32 Debug-----
BoostSmartPtr.exe - 0 error(s), 0 warning(s)

Build > Debug > Find in Files > Find > Ready
```

Ready

48

標準庫版本, Visual C++

files layout

...\\include →



— 侯捷 —

49

標準庫版本, GNU C++

(Dev-C++ 5.11 畫面; with GNU 4.9.2)

The screenshot shows the Dev-C++ 5.11 IDE interface. The title bar reads "C++11 - [C++11.dev] - Dev-C++ 5.11". The menu bar includes File, Edit, Search, View, Project, Execute, Tools, AS-Style, Window, Help. The toolbar has icons for file operations like Open, Save, Build, Run, and Debug. The status bar at the bottom shows "Line: 2161 Col: 1 Set: 0 Lines: 2989 Length: 105275 Insert Done parsing 511 files in 1.281 seconds (398.91 files per second)".

The main window displays the code for "TestC++11.cpp" under the "C++11" project. The code uses `#include <__gnu_cxx/cxxnew.h>` and demonstrates various allocators:

```
//從語法上試用各式各樣的 allocators
cout << sizeof(__gnu_cxx::malloc_allocator<int>) << endl; //1. 大小 1 值其實為 0, fields 都是 static
cout << sizeof(__gnu_cxx::__pool_alloc<int>) << endl; //1
cout << sizeof(__gnu_cxx::__mt_alloc<int>) << endl; //1
cout << sizeof(__gnu_cxx::bitmap_allocator<int>) << endl; //1
cout << sizeof(__gnu_cxx::array_allocator<int>) << endl; //8 ==> 因為它有一個 ptr 指向 array 和一個 size
cout << sizeof(__gnu_cxx::debug_allocator<std::allocator<double>>) << endl; //8
//I cout << sizeof(__gnu_cxx::throw_allocator<int>) << endl; //只有 throw_allocator_base, throw_allocator
//清單容器
list<int, __gnu_cxx::malloc_allocator<int>> list_malloc;
deque<int, __gnu_cxx::debug_allocator<std::allocator<int>>> deque_debug;
vector<int, __gnu_cxx::__pool_alloc<int>> vector_pool;
//測試 cookie
cout << "sizeof(int) = " << sizeof(int) << endl; //4
cout << "sizeof(double) = " << sizeof(double) << endl; //8
cookie_test(std::allocator<int>(), 1); //指派 10h (表示無 cookie)
cookie_test(__gnu_cxx::malloc_allocator<int>(), 1); //指派 10h (表示有 cookie)
//
```

The "Compile Log" tab shows the compilation results:

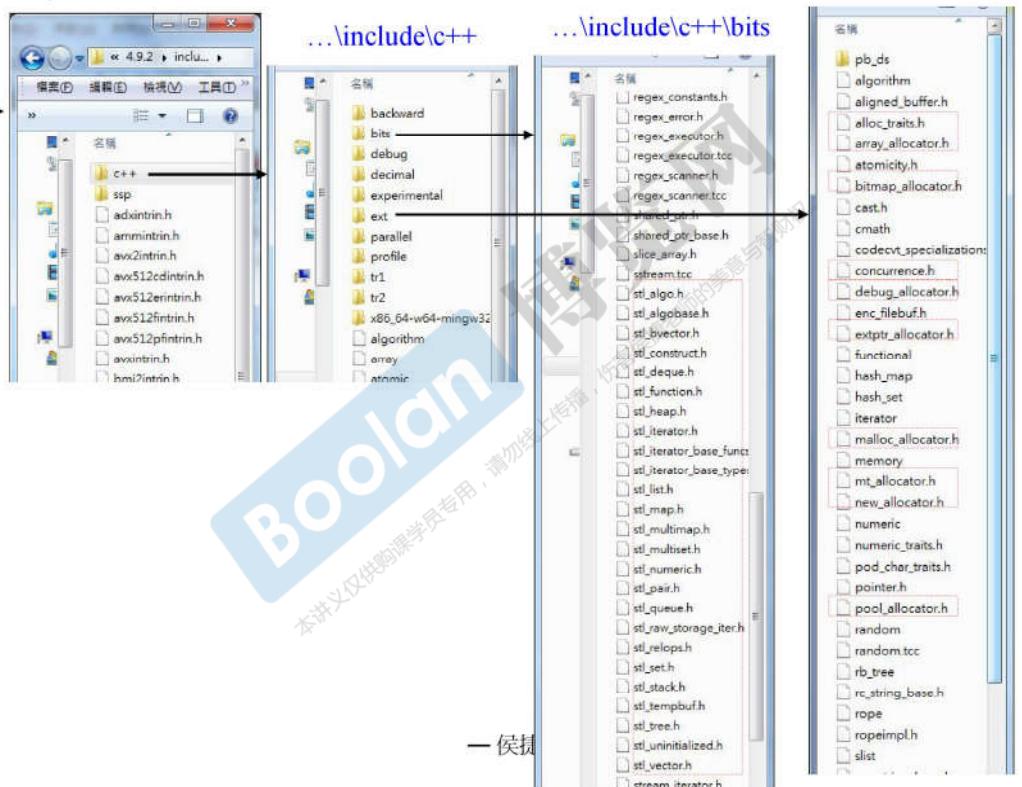
```
g++-exe Test-C++11.o -o C++11.exe -L"C:/Program Files/Dev-Cpp/MinGW64/lib32" -L"C:/Program Files/Dev-Cpp/MinGW64/include" -fno-strict-aliasing
Compiling results...
-----
- Errors: 0
- Warnings: 0
- Output Filename: D:\handout\C++11-test-DevC++\Test-C++11\C++11.exe
- Output Size: 2.1048574476318 MiB
- Compilation Time: 7.05s
```



標準庫版本, GNU C++

files layout

...\\4.9.2\\include →



■■■■ OOP (Object-Oriented programming) vs. GP (Generic Programming)

OOP 企圖將 **datas** 和 **methods** 關聯在一起

```
template <class T,  
         class Alloc = alloc>  
class list {  
...  
    void sort();  
};
```

為什麼 list 不能使用 ::sort() 排序？

::sort(c.begin(), c.end());

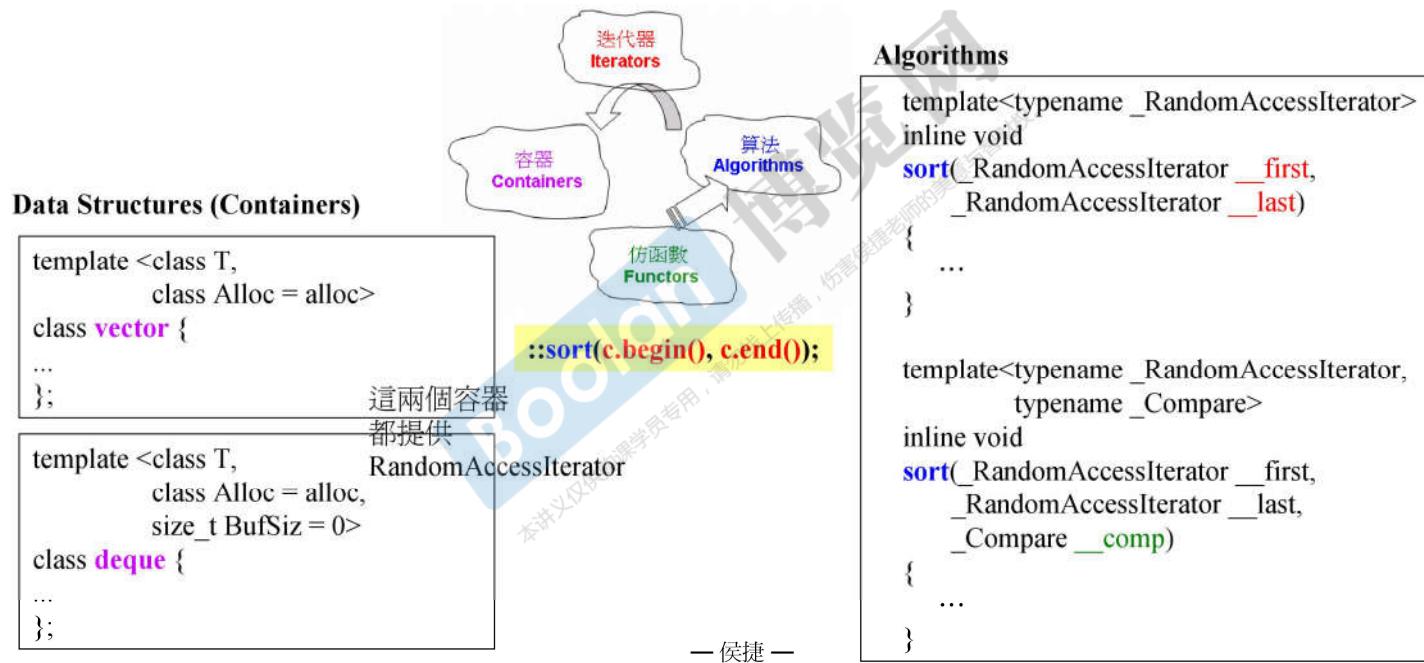
```
template <class RandomAccessIterator>  
inline void sort(RandomAccessIterator first, RandomAccessIterator last) {  
    if (first != last) {  
        __introsort_loop(first, last, value_type(first), __lg(last - first) * 2);  
        __final_insertion_sort(first, last);  
    }  
}
```

```
template <class RandomAccessIterator, class T, class Size>  
void __introsort_loop(RandomAccessIterator first,  
                      RandomAccessIterator last,  
                      T*,  
                      Size depth_limit) {  
...  
    RandomAccessIterator cut = __unguarded_partition  
        (first, last, T(__median(*first, *(first + (last - first)/2), *(last - 1))));  
...  
}
```

只有 RandomAccessIterator
才能如此操作

■■■ OOP (Object-Oriented programming) vs. GP (Generic Programming)

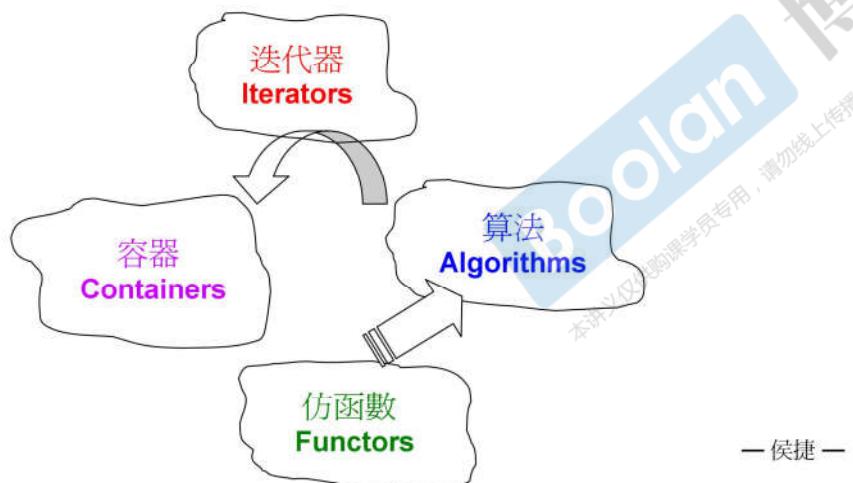
GP 却是將 **datas** 和 **methods** 分開來



■■■■ OOP (Object-Oriented programming) vs. GP (Generic Programming)

採用 GP :

- Containers 和 Algorithms 團隊可各自閉門造車，其間以 Iterator 溝通即可。
- Algorithms 通過 Iterators 確定操作範圍，並通過 Iterators 取用 Container 元素。



— 侯捷 —

```
template <class T>
inline const T& min(const T& a, const T& b) {
    return b < a ? b : a;
}
```

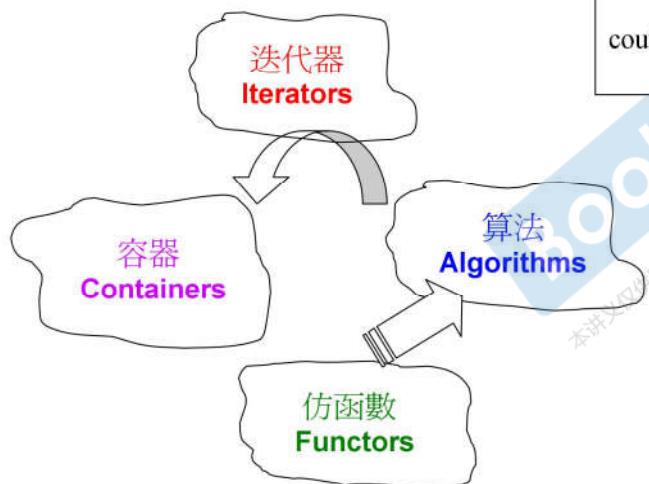
```
template <class T>
inline const T& max(const T& a, const T& b) {
    return a < b ? b : a;
}
```

```
template <class T, class Compare>
inline const T& min(const T& a, const T& b,
                    Compare comp) {
    return comp(b, a) ? b : a;
}
```

```
template <class T, class Compare>
inline const T& max(const T& a, const T& b,
                    Compare comp) {
    return comp(a, b) ? b : a;
}
```

■■■ OOP (Object-Oriented programming) vs. GP (Generic Programming)

所有 **algorithms**，其內最終涉及元素本身的操
作，無非就是比大小。



```
bool  
strLonger(const string& s1,  
          const string& s2)  
{ return s1.size() < s2.size(); }  
  
cout << "max of zoo and hello : "  
     << max(string("zoo"), string("hello")) << endl; //zoo  
  
cout << "longest of zoo and hello : "  
     << max(string("zoo"), string("hello"), strLonger) << endl; //hello
```

```
template <class T>  
inline const T& max(const T& a, const T& b) {  
    return a < b ? b : a;  
}
```

```
template <class T, class Compare>  
inline const T& max(const T& a, const T& b,  
                   Compare comp) {  
    return comp(a, b) ? b : a;  
}
```

— 侯捷 —

■■■ 閱讀 C++ 標準庫 源碼 (source code) 之必要基礎

- **Operator Overloading** 操作符重載
- **Templates** 模板



■■■■ Operator Overloading, 操作符重載

<http://en.cppreference.com/w/cpp/language/operators>

Overloaded operators

When an operator appears in an **expression**, and at least one of its operands has a **class type** or an **enumeration type**, then **overload resolution** is used to determine the user-defined function to be called among all the functions whose signatures match the following:

Expression	As member function	As non-member function	Example
@a	(a).operator@()	operator@(a)	<code>!std::cin calls std::cin.operator!()</code>
a@b	(a).operator@(b)	operator@(a, b)	<code>std::cout << 42 calls std::cout.operator<<(42)</code>
a=b	(a).operator=(b)	cannot be non-member	<code>std::string s; s = "abc"; calls s.operator=("abc")</code>
a[b]	(a).operator[](b)	cannot be non-member	<code>std::map<int, int> m; m[1] = 2; calls m.operator[](1)</code>
a->	(a).operator->()	cannot be non-member	<code>std::unique_ptr<S> ptr(new S); ptr->bar() calls ptr.operator->()</code>
a@	(a).operator@(0)	operator@(a, 0)	<code>std::vector<int>::iterator i = v.begin(); i++ calls i.operator++(0)</code>

In this table, @ is a placeholder representing all matching operators: all prefix operators in @a, all postfix operators other than -> in a@, all infix operators other than = in a@b

■■■■ Operator Overloading, 操作符重載

<http://en.cppreference.com/w/cpp/language/operators>

Overloaded operators (but not the built-in operators) can be called using function notation:

```
std::string str = "Hello, ";
str.operator+=( "world"); // same as str += "world";
operator<<(operator<<(std::cout, str), '\n'); // same as std::cout << str << '\n';
```

Restrictions

- The operators `::` (scope resolution), `.` (member access), `.*` (member access through pointer to member), and `?:` (ternary conditional) cannot be overloaded.
- New operators such as `**`, `<>`, or `&|` cannot be created.
- The overloads of operators `&&`, `||`, and `,` (comma) lose their special properties: short-circuit evaluation and `sequencing`.
- The overload of operator `->` must either return a raw pointer or return an object (by reference or by value), for which operator `->` is in turn overloaded.
- It is not possible to change the precedence, grouping, or number of operands of operators.

■■■■ Operator Overloading, 操作符重载

```
template<class T, class Ref, class Ptr>
struct __list_iterator {
    typedef __list_iterator<T, Ref, Ptr> self;
    typedef bidirectional_iterator_tag iterator_category; // (1)
    typedef T value_type; // (2)
    typedef Ptr pointer; // (3)
    typedef Ref reference; // (4)
    typedef __list_node<T>* link_type;
    typedef ptrdiff_t difference_type; // (5)

    link_type node;

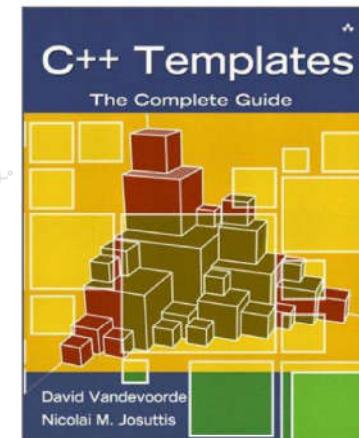
    reference operator*() const { return (*node).data; }
    pointer operator->() const { return &(operator*()); }
    self& operator++()
    self operator++(int)
    ...
},
```

Class Templates, 類模板

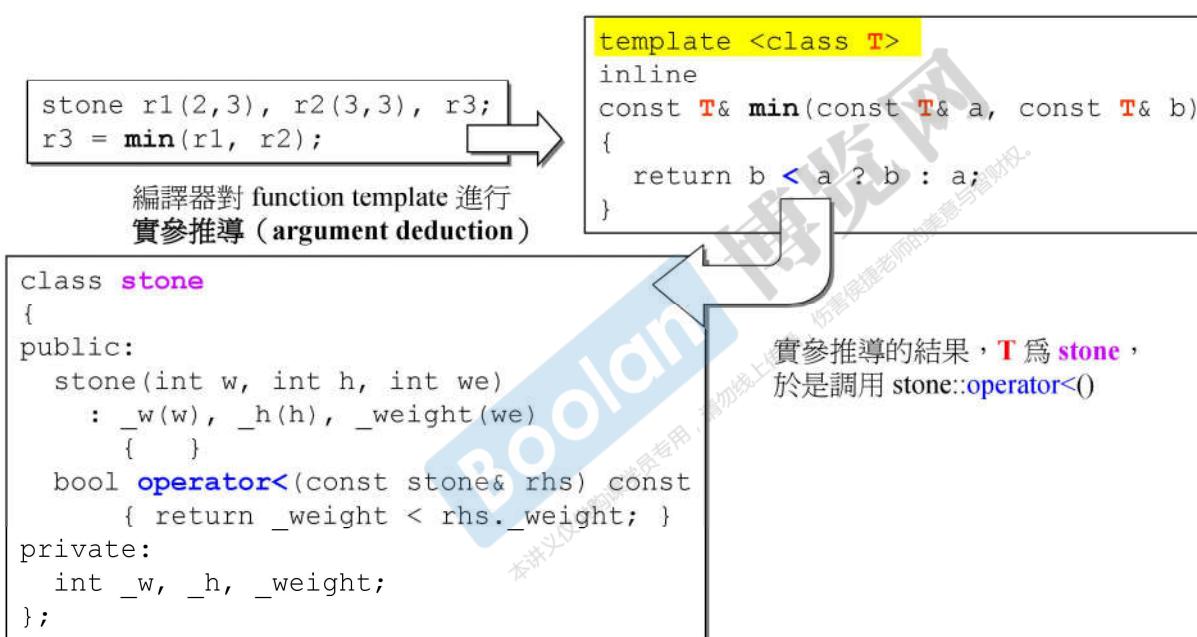
```
template <typename T>
class complex
{
public:
    complex (T r = 0, T i = 0)
        : re (r), im (i)
    { }
    complex& operator += (const complex&);
    T real () const { return re; }
    T imag () const { return im; }
private:
    T re, im;

    friend complex& __doapl (complex*, const complex&);
};
```

```
{
    complex<double> c1(2.5,1.5);
    complex<int> c2(2,6);
    ...
}
```



Function Templates, 函數模板



■■■ Member Templates, 成員模板

```
template <class T1, class T2>
struct pair {
    typedef T1 first_type;
    typedef T2 second_type;

    T1 first;
    T2 second;
    pair() : first(T1()), second(T2()) {}
    pair(const T1& a, const T2& b) : first(a), second(b) {}

#define __STL_MEMBER_TEMPLATES
    template <class U1, class U2>
    pair(const pair<U1, U2>& p) : first(p.first), second(p.second) {}
#endif
};
```

Specialization, 特化

泛化

```
template <class type>
struct __type_traits {
    typedef __true_type
    typedef __false_type
    typedef __false_type
    typedef __false_type
    typedef __false_type
    typedef __false_type
};
```

```
struct __true_type { };
struct __false_type { };
```

ref. G2.91 <type_traits.h>

this_dummy_member_must_be_first;
has_trivial_default_constructor;
has_trivial_copy_constructor;
has_trivial_assignment_operator;
has_trivial_destructor;
is_POD_type;

Plain Old Data

特化

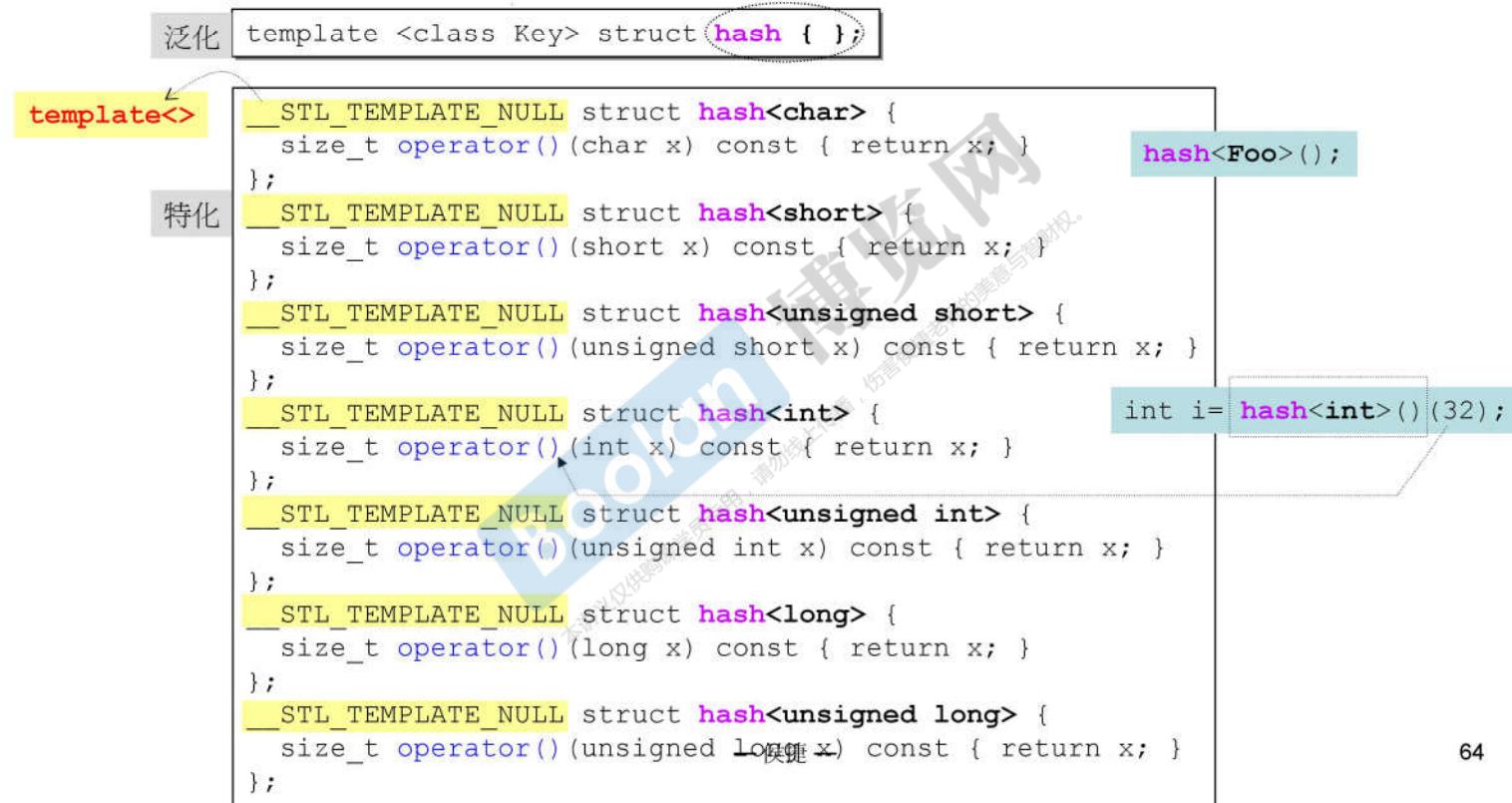
```
template<> struct __type_traits<int> {
    typedef __true_type
    typedef __true_type
    typedef __true_type
    typedef __true_type
    typedef __true_type
};
```

__type_traits<Foo>::has_trivial_destructor

特化

```
template<> struct __type_traits<double> {
    typedef __true_type
    typedef __true_type
    typedef __true_type
    typedef __true_type
    typedef __true_type
};
```

Specialization, 特化



■■■■ Specialization, 特化

```
泛化 template<typename _Tp>
    class allocator;

    /// allocator<void> specialization.
    template<>
        class allocator<void>
    {
    public:
        typedef size_t      size_type;
        typedef ptrdiff_t   difference_type;
        typedef void*       pointer;
        typedef const void* const_pointer;
        typedef void        value_type;

        template<typename _Tp1>
            struct rebind
            { typedef allocator<_Tp1> other; };
    };
}
```

■■■■ Partial Specialization, 偏特化

ref. G2.9 <stl_iterator.h>

泛化

```
template <class T,  
         class Alloc = alloc>  
class vector  
{  
    ...  
};
```

偏特化

```
template<class Alloc>  
class vector<bool, Alloc>  
{  
    ...  
};
```

```
template <class Iterator>  
struct iterator_traits { 泛化  
    typedef typename Iterator::iterator_category iterator_category;  
    typedef typename Iterator::value_type value_type;  
    typedef typename Iterator::difference_type difference_type;  
    typedef typename Iterator::pointer pointer;  
    typedef typename Iterator::reference reference;  
};  
//partial specialization for regular pointers  
template <class T>  
struct iterator_traits<T*> { 偏特化  
    typedef random_access_iterator_tag iterator_category;  
    typedef T value_type;  
    typedef ptrdiff_t difference_type;  
    typedef T* pointer;  
    typedef T& reference;  
};  
//partial specialization for regular const pointers  
template <class T>  
struct iterator_traits<const T*> { 偏特化  
    typedef random_access_iterator_tag iterator_category;  
    typedef T value_type;  
    typedef ptrdiff_t difference_type;  
    typedef const T* pointer;  
    typedef const T& reference;  
};
```

■■■ 分配器 allocators 先談 operator new() 和 malloc()

...\\vc98\\crt\\src\\newop2.cpp

```
void *operator new(size_t size, const std::nothrow_t&)
    _THROW0()
{ // try to allocate size bytes
void *p;
while ((p = malloc(size)) == 0)
    { // buy more memory or return null pointer
    _TRY_BEGIN
        if (_callnewh(size) == 0) break;
    _CATCH(std::bad_alloc) return (0);
    _CATCH_END
    }
return (p);
}
```

<new.h> of CB5

```
// inline versions of the nothrow_t versions of new & delete operators
inline void *_RTLENTRY operator new (size_t size, const std::nothrow_t &)
{
    size = size ? size : 1;
    return malloc(size);
}
```

00000041
00790b30
00000079
0000000c
00000002
00000007
4 個 0xfd
fill 0xcd (ch)
4 個 0xfd
00000000 (pad)
00000000 (pad)
00000041

■■■■ 分配器 allocators

VC6 STL 對 allocator 的使用

```
template<class _Ty, class _A = allocator<_Ty> >
class vector
{ ...
};
```

```
template<class _Ty, class _A = allocator<_Ty> >
class list
{ ...
};
```

```
template<class _Ty, class _A = allocator<_Ty> >
class deque
{ ...
};
```

```
template<class _K, class _Pr = less<_K>,
         class _A = allocator<_K> >
class set {
...
};
```

■■■ 分配器 allocators

VC6 所附的標準庫，其 `allocator` 實現如下 (`<xmemory>`)

```
template<class _Ty>
class allocator {
public:
    typedef _SIZT size_type;
    typedef _PDFT difference_type;
    typedef _Ty _FARQ *pointer;
    typedef _Ty value_type;
    pointer allocate(size_type _N, const void *)
    { return (_Allocate((difference_type)_N, (pointer)0)); }
    void deallocate(void _FARQ * _P, size_type)
    { operator delete(_P); }
};
```

VC6+ 的 allocator 只是以 `::operator new` 和 `::operator delete` 完成 `allocate()` 和 `deallocate()`，沒有任何特殊設計。

其中用到的 `_Allocate()` 定義如下：

```
template<class _Ty>inline
_Ty _FARQ * _Allocate(_PDFT _N, _Ty _FARQ *)
{if (_N < 0) _N = 0;
 return ((_Ty _FARQ *)operator new((_SIZT)_N * sizeof (_Ty))); }
```

//分配 512 ints.
int* p = allocator<int>().allocate(512, (int*)0);
allocator<int>().deallocate(p, 512);

■■■ 分配器 allocators

BC5 STL 對 allocator 的使用

```
template <class T, class Allocator _RWSTD_COMPLEX_DEFAULT(allocator<T>) >
class vector ...

template <class T, class Allocator _RWSTD_COMPLEX_DEFAULT(allocator<T>) >
class list ...

template <class T, class Allocator _RWSTD_COMPLEX_DEFAULT(allocator<T>) >
class deque ...
```

↓
define _RWSTD_COMPLEX_DEFAULT(a) = a <stdcomp.h>

```
template <class T, class Allocator = allocator<T> >
class vector ...

template <class T, class Allocator = allocator<T> >
class list ...

template <class T, class Allocator = allocator<T> >
class deque ...
```

■■■■ 分配器 allocators

BC5 所附的標準庫，其 `allocator` 實現如下 (`<memory.stl>`)

```
template <class T>
class allocator
{
public:
    typedef size_t           size_type;
    typedef ptrdiff_t        difference_type;
    typedef T*                pointer;
    typedef T                value_type;

    pointer allocate(size_type n, allocator<void>::const_pointer = 0) {
        pointer tmp =
            _RWSTD_STATIC_CAST(pointer, (::operator new
                (_RWSTD_STATIC_CAST(size_t, (n * sizeof(value_type))))));
        _RWSTD_THROW_NO_MSG(tmp == 0, bad_alloc);
        return tmp;
    }
    void deallocate(pointer p, size_type) {
        ::operator delete(p);
    } <new.h> of CB5
    ...
};
```

BC++ 的 allocator 只是以 `::operator new` 和 `::operator delete` 完成 `allocate()` 和 `deallocate()`，沒有任何特殊設計。

//分配 512 ints.
`int* p = allocator<int>().allocate(512);
allocator<int>().deallocate(p,512);`

■■■ 分配器 allocators

G2.9 所附的標準庫，其 allocator 實現如下 (<defalloc.h>)

GCC2.9 的 allocator 只是以 ::operator new 和 ::operator delete 完成 allocate() 和 deallocate()，沒有任何特殊設計。

```
template <class T>
class allocator {
public:
    typedef T      value_type;
    typedef T*     pointer;
    typedef size_t size_type;
    typedef ptrdiff_t difference_type;
    pointer allocate(size_type n) {
        return ::allocate((difference_type)n, (pointer)0);
    }
    void deallocate(pointer p) { ::deallocate(p); }
};
```

```
template <class T>
inline T* allocate(ptrdiff_t size, T*) {
    set_new_handler(0);
    T* tmp = (T*)
        ::operator new((size_t)(size*sizeof(T)));
    if (tmp == 0) {
        cerr << "out of memory" << endl;
        exit(1);
    }
    return tmp;
}
template <class T>
inline void deallocate(T* buffer)
    ::operator delete(buffer);
```

G++ <defalloc.h> 中有這樣的註釋：DO NOT USE THIS FILE unless you have an old container implementation that requires an allocator with the HP-style interface. SGI STL uses a different allocator interface. SGI-style allocators are not parametrized with respect to the object type; they traffic in void* pointers. This file is not included by any other SGI STL header.

■■■■ 分配器 allocators

G2.9 STL 對 allocator 的使用

```
template <class T, class Alloc = alloc>
class vector {
    ...
};
```

```
template <class T, class Alloc = alloc>
class list {
    ...
};
```

```
template <class T, class Alloc = alloc,
          size_t BufSiz = 0>
class deque {
    ...
};
```

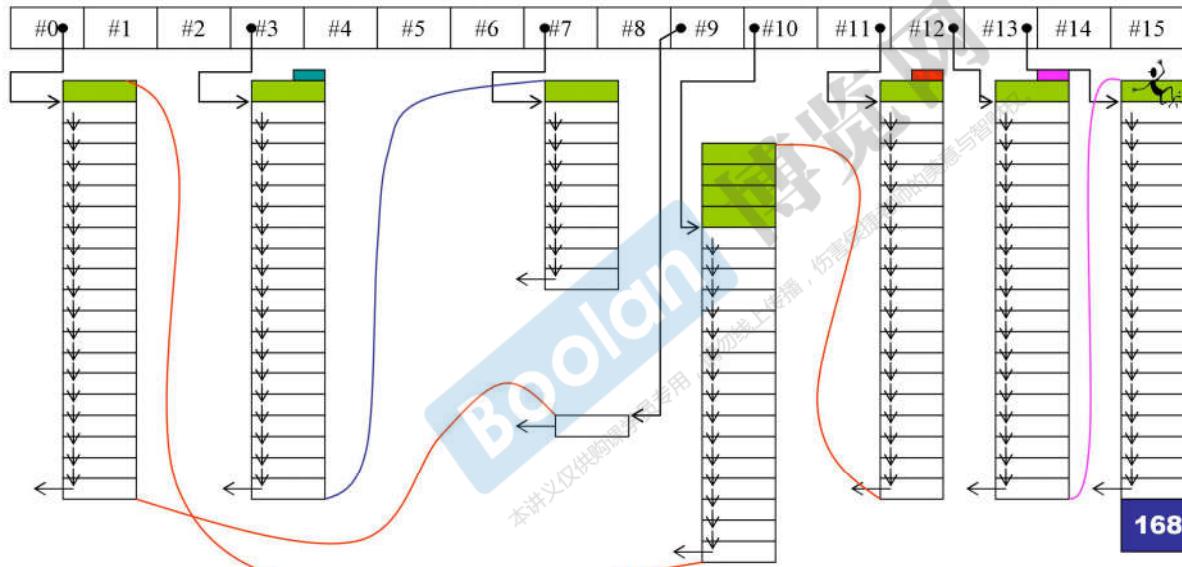
```
//分配 512 bytes.
void* p = alloc::allocate(512); //也可alloc().allocate(512);
alloc::deallocate(p,512);
```

```
template <class Key,
          class T,
          class Compare = less<Key>,
          class Alloc = alloc>
class map {
    ...
};
```

```
template <class Key,
          class Compare = less<Key>,
          class Alloc = alloc>
class set {
    ...
};
```

■■■ 分配器 allocators

G2.9 所附的標準庫，其 `alloc` 實現如下 (`<stl_alloc.h>`)



— 侯捷 —

74

■■■ 分配器 allocators

G4.9 STL 對 allocator 的使用

```
template<typename _Tp, typename _Alloc = std::allocator<_Tp>>
class vector : protected _Vector_base<_Tp, _Alloc>
{
    ...
};
```

■■■■ 分配器 allocators

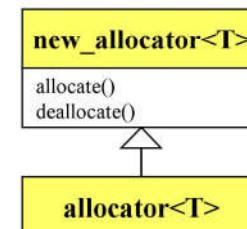
G4.9 所附的標準庫，其 `allocator` 實現如下

```
template<typename _Tp>                                <bits/allocator.h>
class allocator: public __allocator_base<_Tp>
{
    ...
};

# define __allocator_base __gnu_cxx::new_allocator

template<typename _Tp>                                <bits/new_allocator.h>
class new_allocator
{
    ...
    pointer allocate(size_type __n, const void* = 0) {
        if (__n > this->max_size())
            std::__throw_bad_alloc();
        return static_cast<_Tp*>
            (::operator new(__n * sizeof(_Tp)));
    }

    void deallocate(pointer __p, size_type)
    { ::operator delete(__p); }
    ...
};
```



■■■■ 分配器 allocators

G4.9 所附的標準庫，有許多 extention allocators，
其中 `_pool_alloc` 就是 G2.9 的 `alloc`。

用例：

```
vector<string, __gnu_cxx::__pool_alloc<string>> vec;
```

```
template<typename _Tp>
class __pool_alloc : private __pool_alloc_base
{
    ...
};

class __pool_alloc_base
{
protected:
    enum { __S_align = 8 };
    enum { __S_max_bytes = 128 };
    enum { __S_free_list_size = (size_t) __S_max_bytes / (size_t) __S_align };

union __Obj
{
    union __Obj* __M_free_list_link;
    char __M_client_data[1]; // The client sees this.
};

static __Obj* volatile __S_free_list[__S_free_list_size];
...
```

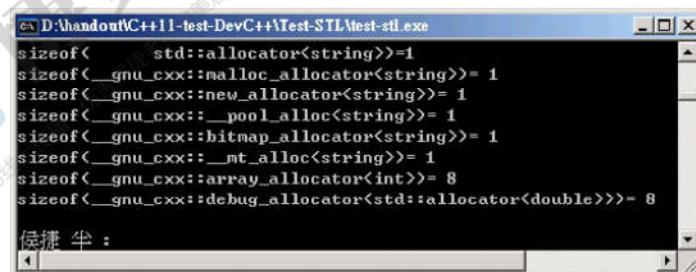
這段有趣的注解仍然沒變。
// Try to make do with what we have. That can't hurt. We
// do not try smaller requests, since that tends to result
// in disaster on multi-process machines.

本讲义仅供课堂学员专用，请勿外传，违者必究。

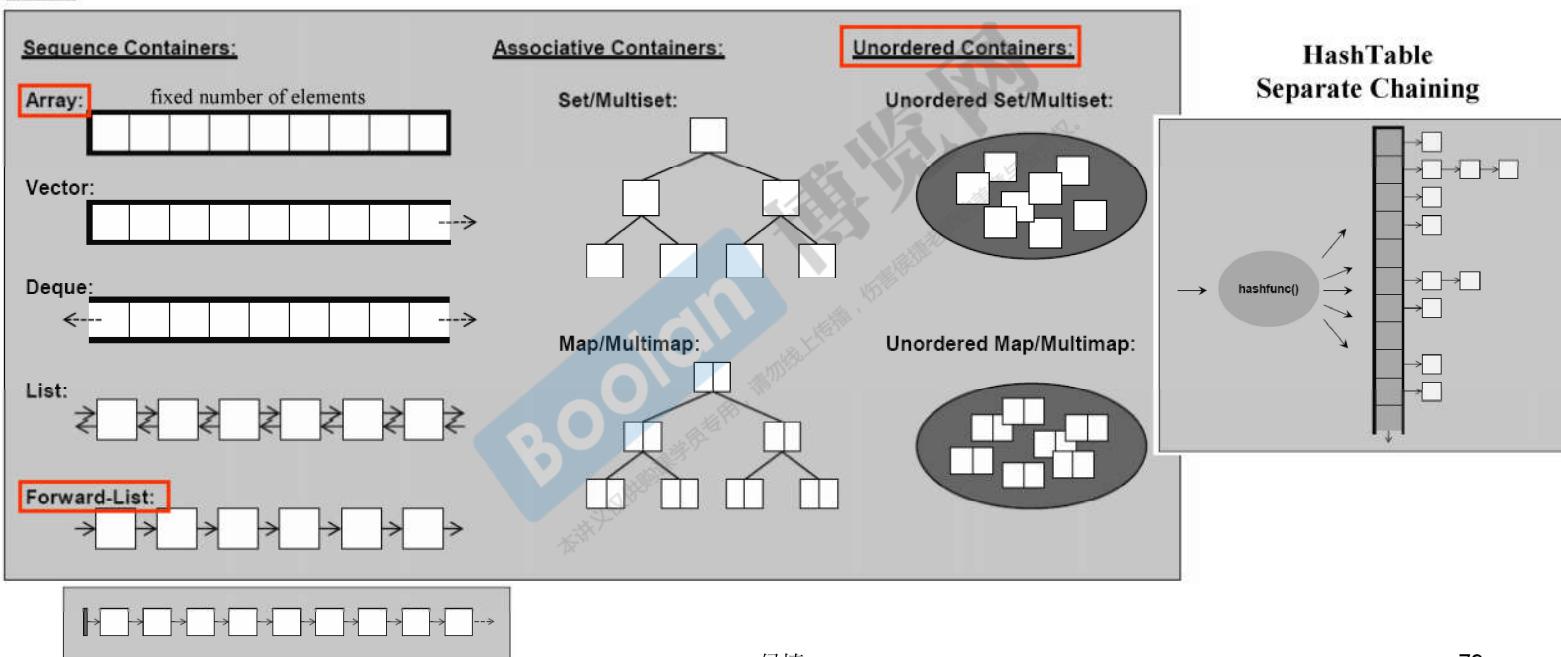
— 侯捷 —

///// 迭代器, sizeof()

```
cout << "sizeof(      std::allocator<string>)=" << sizeof(std::allocator<string>) << endl;           //1 (理論值 0)
cout << "sizeof(_gnu_cxx::malloc_allocator<string>)=" << sizeof(_gnu_cxx::malloc_allocator<string>) << endl; //1 (理論值 0)
cout << "sizeof(_gnu_cxx::new_allocator<string>)=" << sizeof(_gnu_cxx::new_allocator<string>) << endl;        //1 (理論值 0)
cout << "sizeof(_gnu_cxx::_pool_alloc<string>)=" << sizeof(_gnu_cxx::_pool_alloc<string>) << endl;        //1 (理論值 0)
cout << "sizeof(_gnu_cxx::bitmap_allocator<string>)=" << sizeof(_gnu_cxx::bitmap_allocator<string>) << endl; //1 (理論值 0)
cout << "sizeof(_gnu_cxx::_mt_alloc<string>)=" << sizeof(_gnu_cxx::_mt_alloc<string>) << endl;          //1 (理論值 0)
cout << "sizeof(_gnu_cxx::array_allocator<int>)=" << sizeof(_gnu_cxx::array_allocator<int>) << endl;         //8
cout << "sizeof(_gnu_cxx::debug_allocator<std::allocator<double>>)"= << sizeof(_gnu_cxx::debug_allocator<std::allocator<double>>) << endl; //8
//=> 因為它有一個 ptr 指向 array 和一個 size_t 表示消耗到 array 哪兒
```



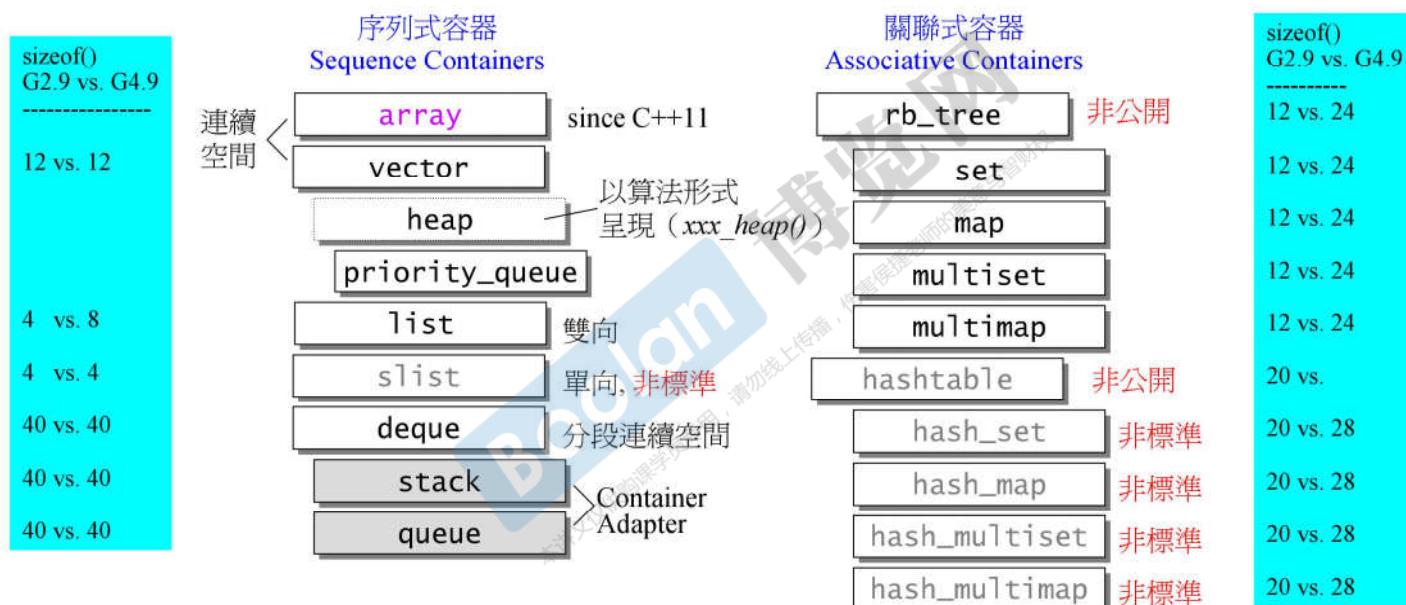
容器 – 結構與分類



— 侯捷 —

容器, 結構與分類

本圖以縮排形式表達“基層與衍生層”的關係。
這裡所謂衍生，並非繼承（inheritance）而是複合（composition）。



在 C++11 中，slist 名為 `forward_list`，`hash_set`, `hash_map` 名為 `unordered_set`, `unordered_map`，
`hash_multiset`, `hash_multimap` 名為 `unordered_multiset`, `unordered_multimap`；且新添 `array`。

容器和其迭代器, sizeof()

```
cout << "sizeof(array<int,100>)= " << sizeof(array<int,100>) << endl; //400
cout << "sizeof(vector<int>)= " << sizeof(vector<int>) << endl; //12
cout << "sizeof(list<int>)= " << sizeof(list<int>) << endl; //8
cout << "sizeof(forward_list<int>)= " << sizeof(forward_list<int>) << endl; //4
cout << "sizeof(deque<int>)= " << sizeof(deque<int>) << endl; //40
cout << "sizeof(stack<int>)= " << sizeof(stack<int>) << endl; //40
cout << "sizeof(queue<int>)= " << sizeof(queue<int>) << endl; //40
cout << "sizeof(set<int>)= " << sizeof(set<int>) << endl; //24
cout << "sizeof(map<int,int>)= " << sizeof(map<int,int>) << endl; //24
cout << "sizeof(multiset<int>)= " << sizeof(multiset<int>) << endl; //24
cout << "sizeof(multimap<int,int>)= " << sizeof(multimap<int,int>) << endl; //24
cout << "sizeof(unordered_set<int>)= " << sizeof(unordered_set<int>) << endl;
cout << "sizeof(unordered_map<int,int>)= " << sizeof(unordered_map<int,int>) << endl;
cout << "sizeof(unordered_multiset<int>)= " << sizeof(unordered_multiset<int>) << endl;
cout << "sizeof(unordered_multimap<int,int>)= " << sizeof(unordered_multimap<int,int>) << endl;

cout << "sizeof(array<int,100>::iterator)= " << sizeof(array<int,100>::iterator) << endl;
cout << "sizeof(vector<int>::iterator)= " << sizeof(vector<int>::iterator) << endl;
cout << "sizeof(list<int>::iterator)= " << sizeof(list<int>::iterator) << endl;
cout << "sizeof(forward_list<int>::iterator)= " << sizeof(forward_list<int>::iterator) << endl;
cout << "sizeof(deque<int>::iterator)= " << sizeof(deque<int>::iterator) << endl; //Er
// cout << "sizeof(stack<int>::iterator)= " << sizeof(stack<int>::iterator) << endl; //Er
// cout << "sizeof(queue<int>::iterator)= " << sizeof(queue<int>::iterator) << endl; //Er
cout << "sizeof(set<int>::iterator)= " << sizeof(set<int>::iterator) << endl;
cout << "sizeof(map<int,int>::iterator)= " << sizeof(map<int,int>::iterator) << endl;
cout << "sizeof(multiset<int>::iterator)= " << sizeof(multiset<int>::iterator) << endl;
cout << "sizeof(multimap<int,int>::iterator)= " << sizeof(multimap<int,int>::iterator) << endl; //4
cout << "sizeof(unordered_set<int>::iterator)= " << sizeof(unordered_set<int>::iterator) << endl; //4
cout << "sizeof(unordered_map<int,int>::iterator)= " << sizeof(unordered_map<int,int>::iterator) << endl; //4
cout << "sizeof(unordered_multiset<int>::iterator)= " << sizeof(unordered_multiset<int>::iterator) << endl; //4
cout << "sizeof(unordered_multimap<int,int>::iterator)= " << sizeof(unordered_multimap<int,int>::iterator) << endl; //4
```

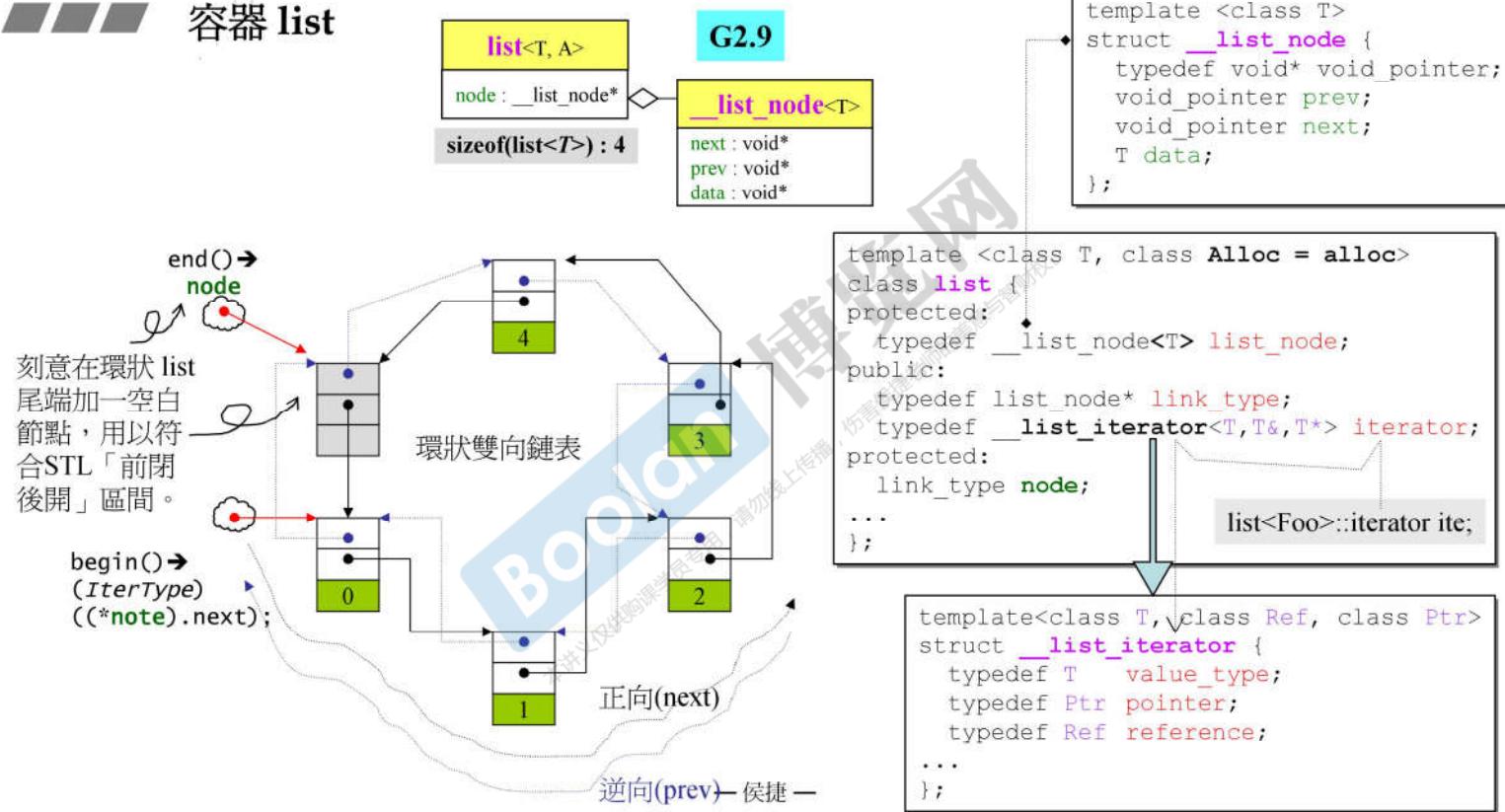
D:\handout\C++11-test\DevC++\Test-STL\test-stl.exe

```
test_components_size().....
sizeof(array<int,100>)= 400
sizeof(vector<int>)= 12
sizeof(list<int>)= 8
sizeof(forward_list<int>)= 4
sizeof(deque<int>)= 40
sizeof(stack<int>)= 40
sizeof(queue<int>)= 40
sizeof(set<int>)= 24
sizeof(map<int,int>)= 24
sizeof(multiset<int>)= 24
sizeof(multimap<int,int>)= 24
sizeof(unordered_set<int>)= 28
sizeof(unordered_map<int,int>)= 28
sizeof(unordered_multiset<int>)= 28
sizeof(unordered_multimap<int,int>)= 28
sizeof(array<int,100>::iterator)= 4
sizeof(vector<int>::iterator)= 4
sizeof(list<int>::iterator)= 4
sizeof(forward_list<int>::iterator)= 4
sizeof(deque<int>::iterator)= 16
sizeof(stack<int>::iterator)= 4
sizeof(queue<int>::iterator)= 4
sizeof(set<int>::iterator)= 4
sizeof(map<int,int>::iterator)= 4
sizeof(multiset<int>::iterator)= 4
sizeof(multimap<int,int>::iterator)= 4
sizeof(unordered_set<int>::iterator)= 4
sizeof(unordered_map<int,int>::iterator)= 4
sizeof(unordered_multiset<int>::iterator)= 4
sizeof(unordered_multimap<int,int>::iterator)= 4
```

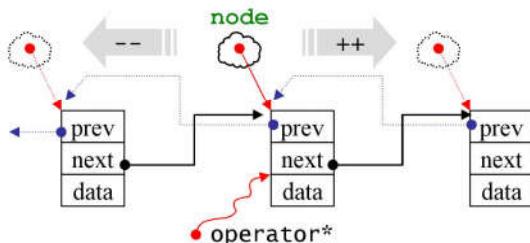
G4.9

81

容器 list



list's iterator



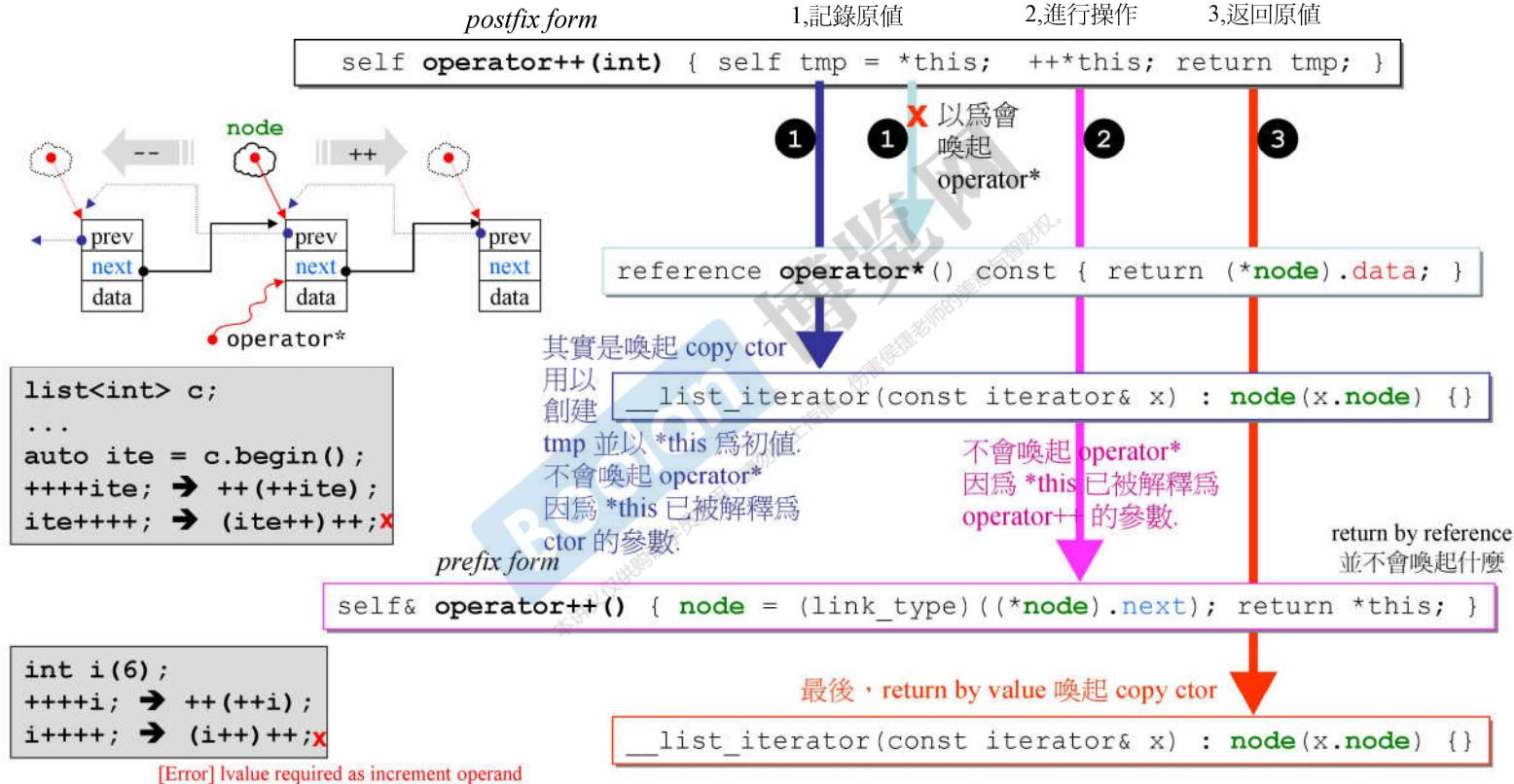
```
template <class T>
struct __list_node {
    typedef void* void_pointer;
    void_pointer prev;
    void_pointer next;
    T data;
};
```

```
template<class T, class Ref, class Ptr>
struct __list_iterator {
    typedef __list_iterator<T, Ref, Ptr> self;
    typedef bidirectional_iterator_tag iterator_category; // (1)
    typedef T value_type; // (2)
    typedef Ptr pointer; // (3)
    typedef Ref reference; // (4)
    typedef __list_node<T>* link_type;
    typedef ptrdiff_t difference_type; // (5)

    link_type node;

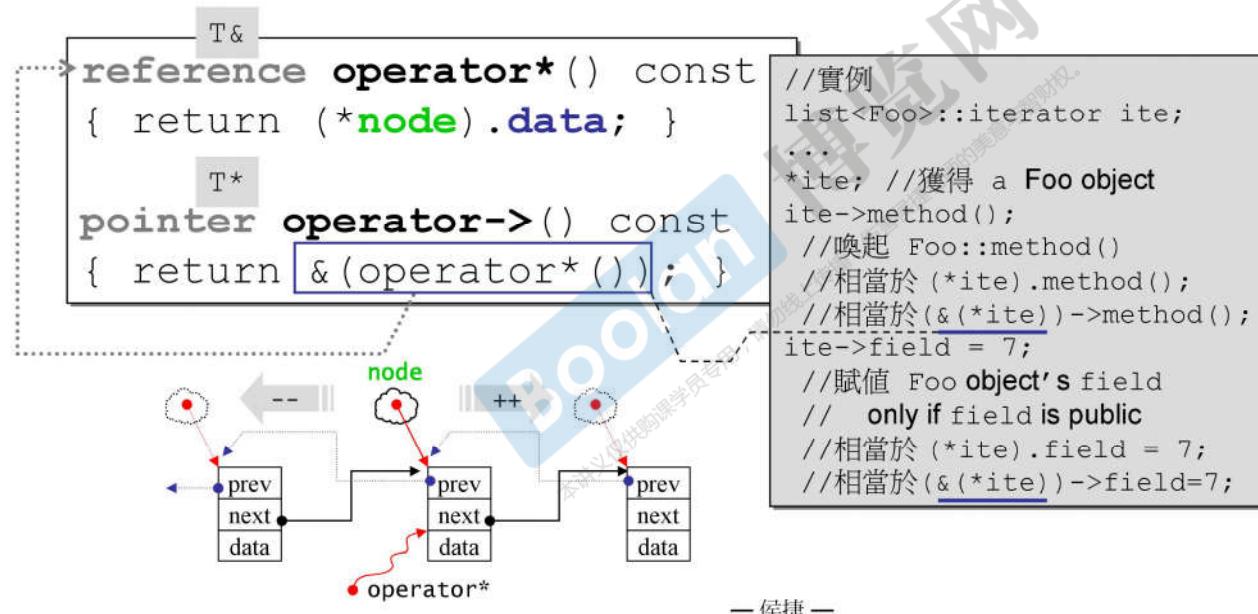
    reference operator*() const { return (*node).data; }
    pointer operator->() const { return &(operator*()); }
    self& operator++()
    { node = (link_type)((*node).next); return *this; }
    self operator++(int)
    { self tmp = *this; ++*this; return tmp; }
    ...
};
```

list's iterator



list's iterator

當你對某個 type 實施 `operator->`，而該 type 並非 built-in ptr 時，編譯器會做一件很有趣的事：在找出 user-defined `operator->` 並將它施行於該 type 後，編譯器會對執行結果再次施行 `operator->`。編譯器不斷執行這動作直至觸及 a pointer to a built-in type，然後才進行成員存取。



— 侯捷 —

list's iterator

```
template <class T,  
         class Alloc = alloc>  
class list {  
public:  
    typedef __list_iterator<T,T&,T*> iterator;  
    ...  
};
```

G2.9

```
template<class T,  
        class Ref, class Ptr>  
struct __list_iterator  
{  
    ...  
    typedef Ptr pointer; // (3)  
    typedef Ref reference; // (4)  
    ...  
};
```

```
template <class T>  
struct __list_node {  
    ...  
    void* void_pointer;  
    void_pointer prev;  
    void_pointer next;  
    T data;  
};
```

```
template<typename _Tp,  
        typename _Alloc = std::allocator<_Tp>>  
class list : protected __List_base<_Tp, _Alloc>  
{  
public: typedef __List_iterator<_Tp> iterator;  
    ...  
};
```

G4.9

```
template<typename _Tp>  
struct __List_iterator  
{  
    ...  
    _Tp* pointer; // (3)  
    _Tp& reference; // (4)  
    ...  
};
```

```
struct __List_node_base  
{  
    ...  
    __List_node_base* _M_next;  
    __List_node_base* _M_prev;  
};
```

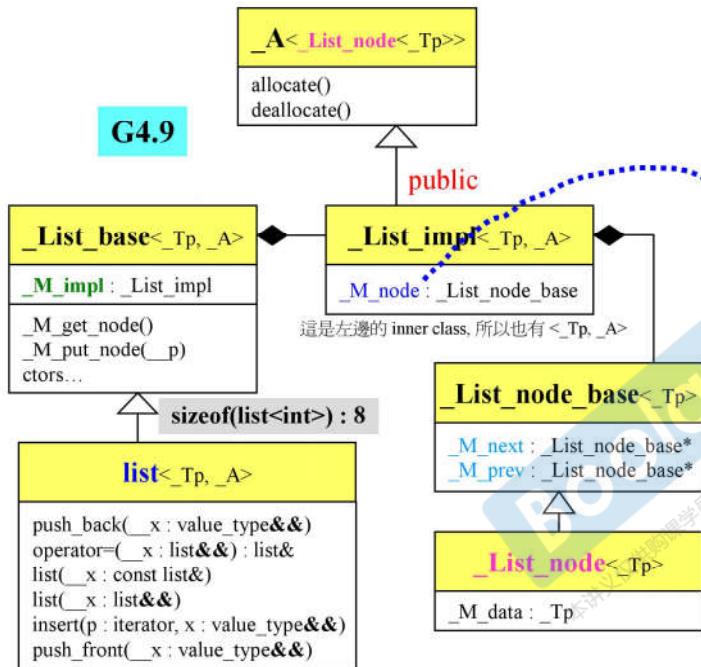
↑

```
template<typename _Tp>  
struct __List_node  
    : public __List_node_base  
{  
    _Tp _M_data;  
};
```

G4.9 相較於 G2.9 :

- 模板參數只有一個 (易理解)
- node 結構有其 parent
- node 的成員的 type 較精確
— 快捷 —

容器 list



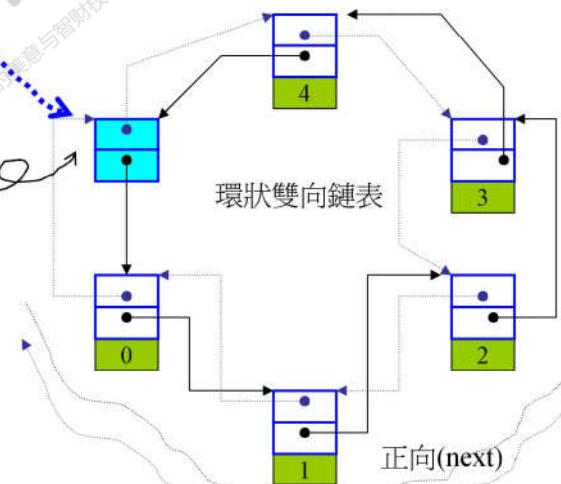
```

iterator
begin() _GLIBCXX_NOEXCEPT
{ return iterator(this->_M_impl._M_node._M_next); }

iterator
end() _GLIBCXX_NOEXCEPT
{ return iterator(&this->_M_impl._M_node); }

```

刻意在環狀 list
尾端加一空白
節點，用以符
合STL「前閉
後開」區間。



— 侯捷 —

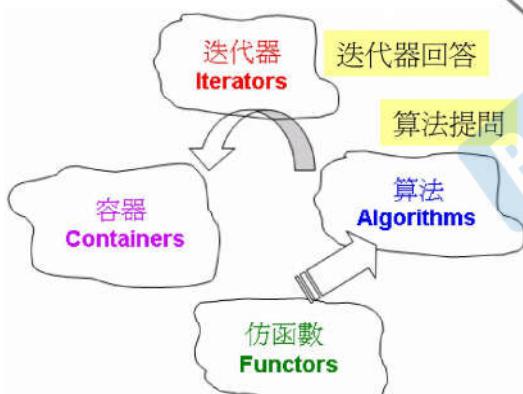
87

Iterator 需要遵循的原則

```
template<typename _ForwardIterator>
inline void
rotate(_ForwardIterator __first,
       _ForwardIterator __middle,
       _ForwardIterator __last)
{
    ...
    std::rotate(__first, __middle, __last,
               std::iterator_category(__first));
}
```

這張圖可看出，
rotate() 需要知道
iterators 的三個
associated types

```
/*
 * This function is not a part of the C++ standard but is syntactic
 * sugar for internal library use only.
 */
template<typename _Iter>
inline typename iterator_traits<_Iter>::iterator_category
__iterator_category(const _Iter&)
{ return typename iterator_traits<_Iter>::iterator_category(); }
```



迭代器回答
Algorithm Answer

// This is a helper function for the rotate algorithm.

```
template<typename _RandomAccessIterator>
void
__rotate(_RandomAccessIterator __first,
         _RandomAccessIterator __middle,
         _RandomAccessIterator __last,
         random_access_iterator_tag)
```

iterators 必須有能力回答
algorithms 的提問

```
...
typedef typename iterator_traits<_RandomAccessIterator>::difference_type _Distance;
typedef typename iterator_traits<_RandomAccessIterator>::value_type _ValueType;
_Distance __n = __last - __first;
_Distance __k = __middle - __first;
...
for (;;) {
    if (__k < __n - __k) {
        if (__is_pod(_ValueType) && __k == 1) {
            _ValueType __t = __GLIBCXX_MOVE(*__p)
```

這樣的提問在 C++ 標準庫
開發過程中設計出 5 種，
本例出現 3 種。另兩種從
未在 C++ 標準庫中被使用
過：reference 和 pointer。

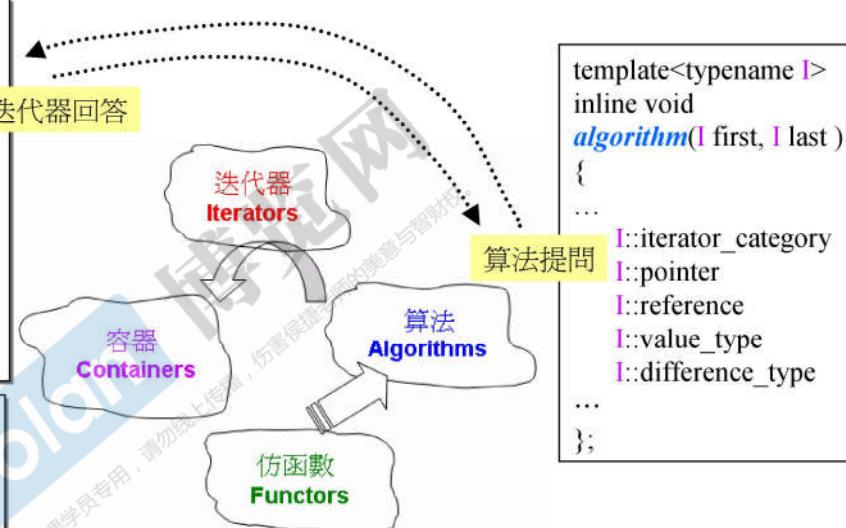
■■■■ Iterator 必須提供的 5 種 associated types

```
template<class T, class Ref, class Ptr>
struct __list_iterator
{
    typedef bidirectional_iterator_tag iterator_category; // (1)
    typedef T      value_type;           // (2)
    typedef Ptr   pointer;             // (3)
    typedef Ref   reference;           // (4)
    typedef ptrdiff_t difference_type; // (5)
    ...
};
```

G2.9

```
template<typename _Tp>
struct _List_iterator
{
    typedef std::bidirectional_iterator_tag iterator_category;
    typedef _Tp      value_type;
    typedef _Tp*   pointer;
    typedef _Tp&   reference;
    typedef ptrdiff_t difference_type;
    ...
};
```

G4.9



```
template<typename I>
inline void
algorithm(I first, I last )
{
    ...
I::iterator_category
I::pointer
I::reference
I::value_type
I::difference_type
...
};
```

但，如果 iterator 並不是個 class 呢？
例如 native pointer
(它被視為一種退化的 iterator)。

— 侯捷 —

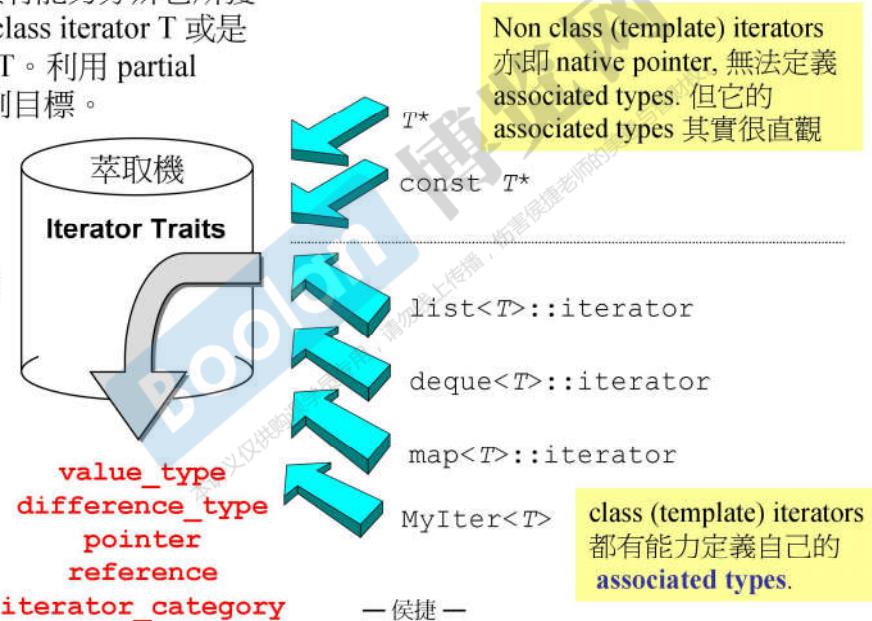
89

■■■ Traits 特性,特徵,特質

Iterator Traits 用以分離 class iterators 和 non-class iterators

這個 traits 機器必須有能力分辨它所獲得的 iterator 是 (1) class iterator T 或是 (2) native pointer to T。利用 partial specialization 可達到目標。

解決計算機問題的尚方寶劍：加一個中介層



— 侯捷 —

/// iterator_traits

Iterator Traits 用以分離 class iterators 和 non-class iterators

```
1 template <class I>
  struct iterator_traits { //traits 是特性之意
    typedef typename I::value_type value_type; //如果 I 是 class iterator 進入這裡
  };

2 //兩個 partial specialization :
template <class T>
struct iterator_traits<T*> {
  typedef T value_type; //如果 I 是 pointer to T 進入這裡
};

3 template <class T>
struct iterator_traits<const T*> {
  typedef T value_type; //注意是 T 而不是 const T //如果 I 是 pointer to const T 進入這裡
};
```

於是當需要知道 I 的 value type 時便可這麼寫：

```
template<typename I,...>
void algorithm(...) {
  typename iterator_traits<I>::value_type v1; //候選
```

value_type 的主要目的是用來聲明變量，而聲明一個無法被賦值的變量沒什麼用，所以 iterator (即便是 constant iterator) 的 value type 不應加上 const。iterator 若是 const int*，其 value_type 應該是 int 而非 const int.

|||| 完整的 iterator_traits

ref G2.9 <stl_iterator.h>

```
template <class I>
struct iterator_traits {
    typedef typename I::iterator_category iterator_category;
    typedef typename I::value_type value_type;
    typedef typename I::difference_type difference_type;
    typedef typename I::pointer pointer;
    typedef typename I::reference reference;
};

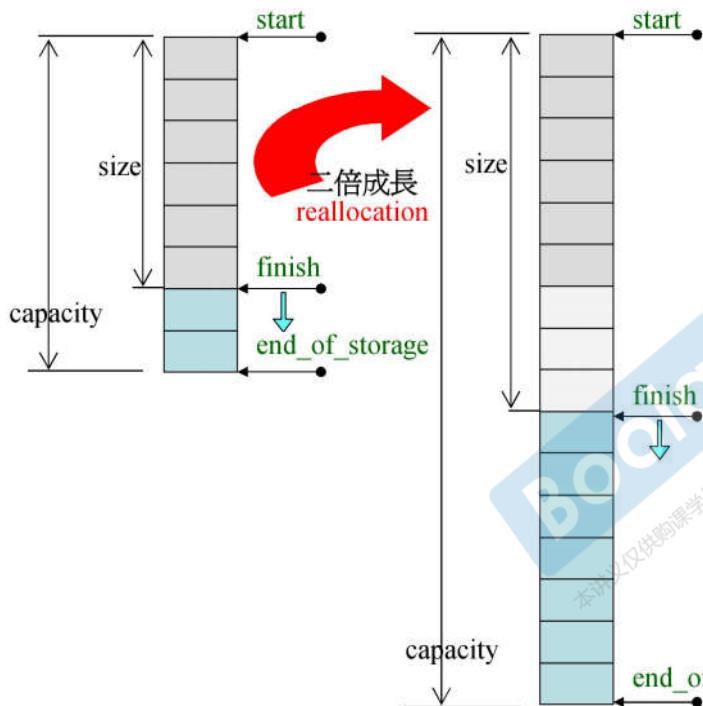
//partial specialization for regular pointers
template <class T>
struct iterator_traits<T*> {
    typedef random_access_iterator_tag iterator_category;
    typedef T value_type;
    typedef ptrdiff_t difference_type;
    typedef T* pointer;
    typedef T& reference;
};

//partial specialization for regular const pointers
template <class T>
struct iterator_traits<const T*> {
    typedef random_access_iterator_tag iterator_category;
    typedef T value_type;
    typedef ptrdiff_t difference_type;
    typedef const T* pointer;
    typedef const T& reference;
};
```

■■■ 各式各樣的 Traits

- **type traits** <.../C++/type_traits>
- **iterator traits** <.../C++/bits/stl_iterator.h>
- **char traits** <.../C++/bits/char_traits.h>
- **allocator traits** <.../C++/bits/alloc_traits.h>
- **pointer traits** <.../C++/bits/ptr_traits.h>
- **array traits** <.../C++/array>

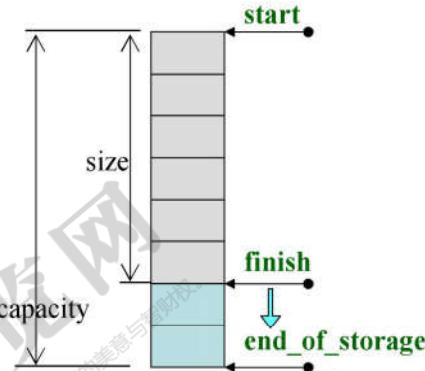
容器 vector



```
template <class T, class Alloc = alloc>
class vector {
public:
    typedef T           value_type;
    typedef value_type* iterator; //T*
    typedef value_type& reference;
    typedef size_t      size_type;
protected:
    iterator start;
    iterator finish;
    iterator end_of_storage;
public:
    iterator begin() { return start; }
    iterator end() { return finish; }
    size_type size() const
    { return size_type(end() - begin()); }
    size_type capacity() const
    { return size_type(end_of_storage - begin()); }
    bool empty() const { return begin() == end(); }
    reference operator[](size_type n)
    { return *(begin() + n); }
    reference front() { return *begin(); }
    reference back() { return *(end() - 1); }
};
```

容器 vector

```
void push_back(const T& x) {
    if (finish != end_of_storage) { //尚有備用空間
        construct(finish, x);      //全局函數
        ++finish;                  //調整水位高度
    }
    else //已無備用空間
        insert_aux(end(), x);
}
```



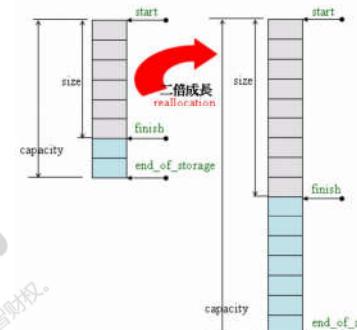
```
template <class T, class Alloc>
void vector<T,Alloc>::insert_aux(iterator position, const T& x) {
    if (finish != end_of_storage) { //尚有備用空間
        //在備用空間起始處建構一個元素，並以vector最後一個元素值為其初值。
        construct(finish, *(finish - 1));
        ++finish; //調整水位。
        T x_copy = x;
        copy_backward(position, finish - 2, finish - 1);
        *position = x_copy;
    }
    else { //已無備用空間
        繼下頁
    }
}
```

容器 vector

```
else { //已無備用空間
    const size_type old_size = size();
    const size_type len = old_size != 0 ? [2 * old_size] : 1;
    //以上分配原則：如果原大小為0，則分配 1 (個元素大小)；  

    //如果原大小不為0，則分配原大小的兩倍，  

    //前半段用來放置原數據，後半段準備用來放置新數據。
    iterator new_start = data_allocator::allocate(len);
    iterator new_finish = new_start;
    try {
        ...
    } catch(...) {
        ...
    }
    //解構並釋放原 vector
    destroy(begin(), end());
    deallocate();
    //調整迭代器，指向新vector
    start = new_start;
    finish = new_finish;
    end_of_storage = new_start + len;
}
try {
    //將原vector的內容拷貝到新vector
    new_finish = uninitialized_copy(start,
                                    position, new_start);
    construct(new_finish, x); //為新元素設初值x
    ++new_finish; //調整水位。
    //拷貝安插點後的原內容（因它也可能被insert(p,x)呼叫）
    new_finish = uninitialized_copy(position,
                                    finish, new_finish);
}
catch(...) {
    //"commit or rollback" semantics.
    destroy(new_start, new_finish);
    data_allocator::deallocate(new_start, len);
    throw;
}
```



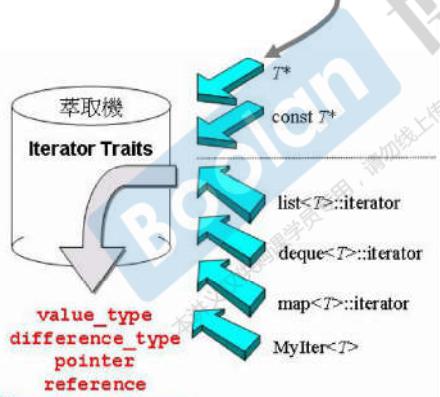
vector's iterator

```
template <class T, class Alloc = alloc>
class vector {
public:
    typedef T value_type;
    typedef value_type* iterator; //T*
    ...
};
```

G2.9

```
vector<int> vec;
...
vector<int>::iterator ite
= vec.begin();
```

- ① iterator_traits<ite>::iterator_category
- ② iterator_traits<ite>::difference_type
- ③ iterator_traits<ite>::value_type



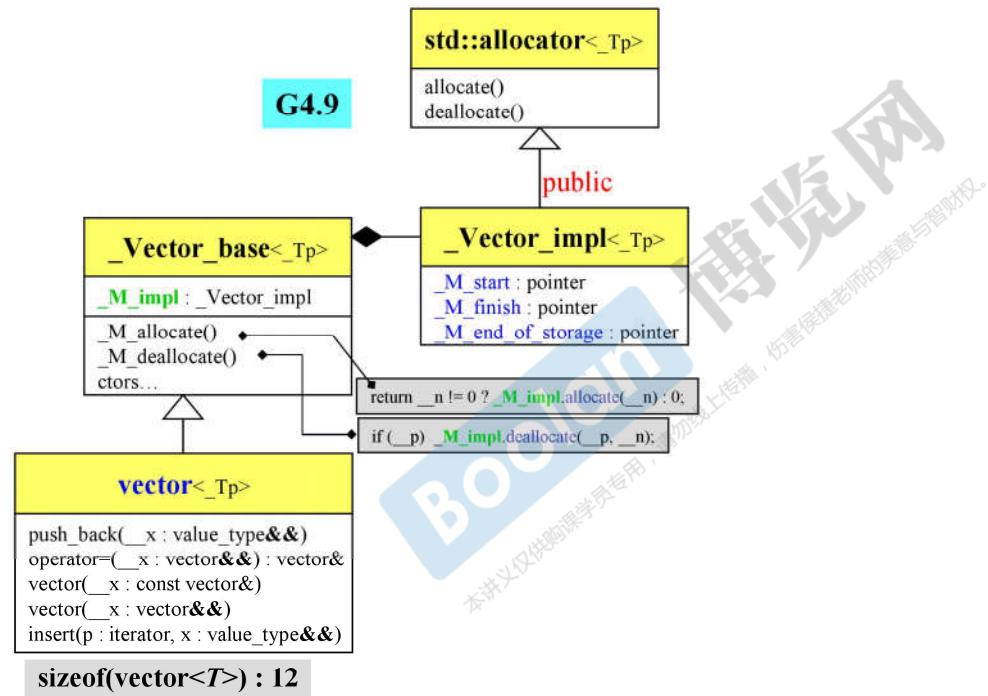
```
template <class I>
struct iterator_traits {
    typedef typename I::iterator_category iterator_category;
    typedef typename I::value_type      value_type;
    typedef typename I::difference_type difference_type;
    typedef typename I::pointer        pointer;
    typedef typename I::reference     reference;
};

//partial specialization for regular pointers
template <class T>
struct iterator_traits<T*> {
    typedef random_access_iterator_tag iterator_category;
    typedef T      value_type;
    typedef ptrdiff_t difference_type;
    typedef T*    pointer;
    typedef T&    reference;
};

//partial specialization for regular const pointers
template <class T>
struct iterator_traits<const T*> {
    typedef random_access_iterator_tag iterator_category;
    typedef T      value_type;
    typedef ptrdiff_t difference_type;
    typedef const T* pointer;
    typedef const T& reference;
};
```

— 侯捷 —

容器 vector



— 侯捷 —

vector's iterator

```
template<typename _Tp, typename _Alloc = std::allocator<_Tp>>
class vector : protected _Vector_base<_Tp, _Alloc>
{
...
typedef _Vector_base<_Tp, _Alloc> _Base;
typedef typename _Base::pointer pointer;
typedef _gnu_cxx::normal_iterator<pointer, vector> iterator;
```

G4.9

```
using std::iterator_traits;
using std::iterator;
template<typename _Iterator, typename _Container>
class _normal_iterator
{
protected:
    _Iterator _M_current;
public:
    typedef _Iterator iterator_type;
    typedef typename _traits_type::iterator_category iterator_category;
    typedef typename _traits_type::value_type value_type;
    typedef typename _traits_type::difference_type difference_type;
    typedef typename _traits_type::reference reference;
    typedef typename _traits_type::pointer pointer;
```

_GLIBCXX_CONSTEXPR **_normal_iterator()** _GLIBCXX_NOEXCEPT
: **_M_current**(**_Iterator**()) {}

本頁追蹤 **vector::iterator**，發現它就是 **_Tp*** 外覆一個 iterator adapter，使能支持 5 associated types

```
template<typename _Tp, typename _Alloc>
struct _Vector_base
{
    ...
    typedef typename _gnu_cxx::alloc_traits<_Alloc>::template
        rebind<_Tp>::other _Tp_alloc_type;
    typedef typename _gnu_cxx::alloc_traits<_Tp_alloc_type>::pointer pointer
```

vector<T>::iterator

_M_current : **_Tp***

```
template<typename _Tp>
class allocator: public _glibcxx_base_allocator<_Tp>
{
public:
    ...
    typedef size_t size_type;
    typedef ptrdiff_t difference_type;
    typedef _Tp* pointer;
    typedef const _Tp* const_pointer;
    typedef _Tp& reference;
    typedef const _Tp& const_reference;
    typedef _Tp value_type;
    ...
    // Inherit everything else.
};
```

9

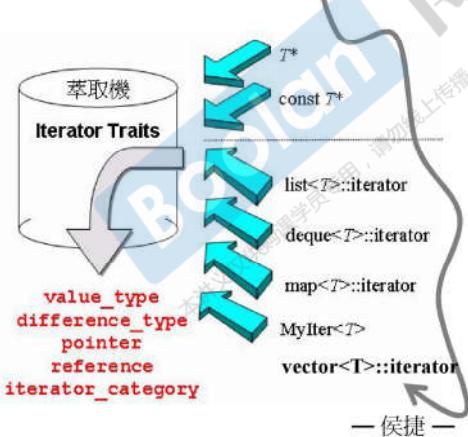
vector's iterator

```
template<typename _Tp, typename _Alloc = std::allocator<_Tp>>
class vector : protected _Vector_base<_Tp, _Alloc>
{
...
typedef _Vector_base<_Tp, _Alloc> _Base;
typedef typename _Base::pointer pointer;
typedef __gnu_cxx::__normal_iterator<pointer, vector> iterator;
```

G4.9

```
vector<int> vec;
...
vector<int>::iterator ite
= vec.begin();
```

- ① **iterator_traits<ite>::iterator_category**
- ② **iterator_traits<ite>::difference_type**
- ③ **iterator_traits<ite>::value_type**



本頁追蹤 **vector::iterator**，發現它就是 **_Tp*** 外覆一個 iterator adapter，使能支持 5 associated types

vector<T>::iterator
_M_current : _Tp*

```
template<typename _Iterator, typename _Container>
class _normal_iterator
{
protected:
    _Iterator _M_current;
    typedef iterator_traits<_Iterator> _traits_type;
public:
    typedef _Iterator iterator_type;
    typedef typename _traits_type::iterator_category iterator_category;
    typedef typename _traits_type::value_type value_type;
    typedef typename _traits_type::difference_type difference_type;
    typedef typename _traits_type::reference reference;
    typedef typename _traits_type::pointer pointer;
    ...
}
```

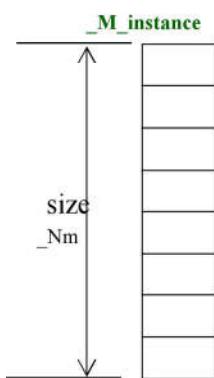
亂七八糟，捨近求遠，何必如此 !!

容器 array

TR1

array<_Tp, _Nm>

_M_instance : _Tp[_Nm]



```
template<typename _Tp, std::size_t _Nm>
struct array
{
    typedef _Tp           value_type;
    typedef _Tp*          pointer;
    typedef value_type*   iterator;    其 iterator 是 native pointer,
                                     (G2.9 vector 也是如此)

    // Support for zero-sized arrays mandatory.
    value_type _M_instance[_Nm ? _Nm : 1];

    iterator begin()
    { return iterator(&_M_instance[0]); }

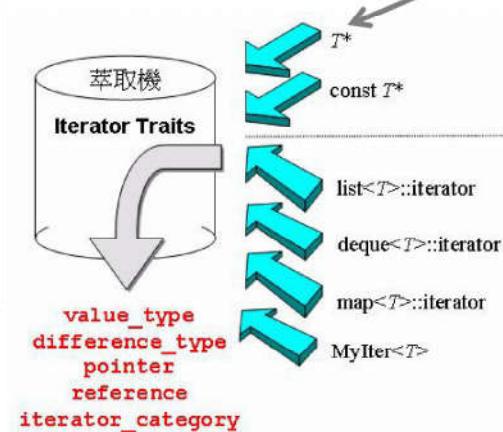
    iterator end()
    { return iterator(&_M_instance[_Nm]); }

    ...
}
```

沒有ctor, 沒有dtor

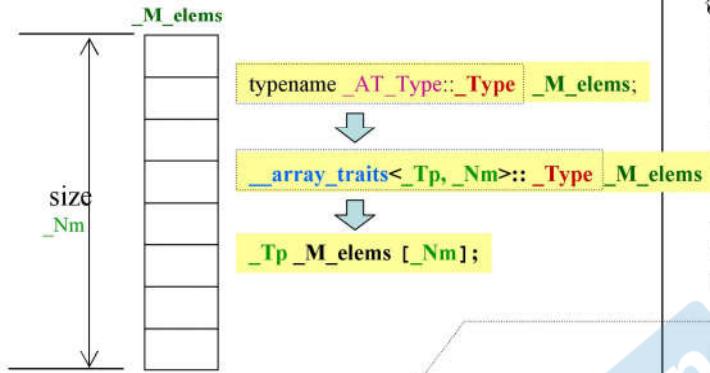
— 侯捷 —

```
array<int, 10> myArray;
auto ite = myArray.begin();
// array<int, 10>::iterator ite = ...
ite += 3;
cout << *ite;
```



容器 array

G4.9



```

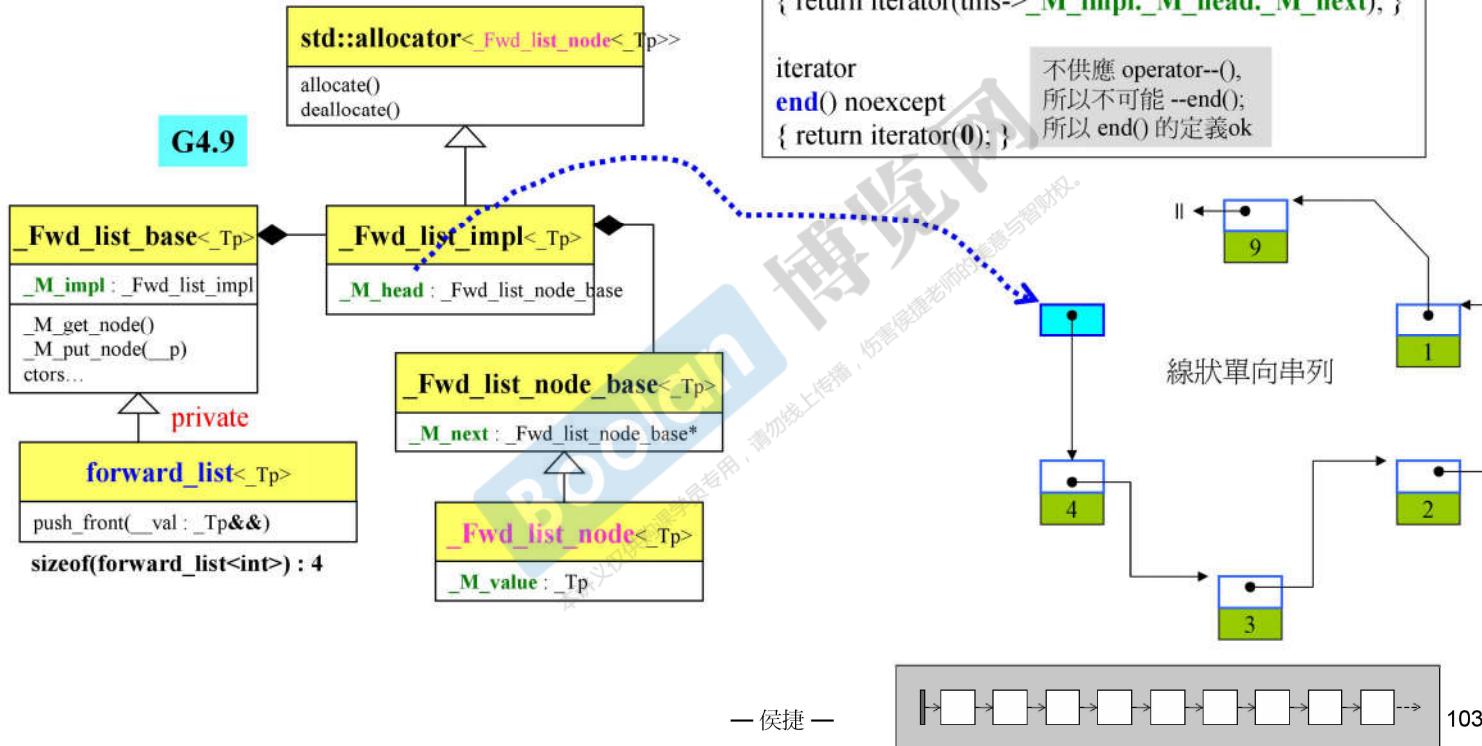
template<typename _Tp, std::size_t _Nm>
struct array
{
    typedef _Tp           value_type;
    typedef value_type*   pointer;
    typedef value_type&  reference;
    typedef value_type*   iterator;
    typedef std::size_t   size_type;

    // Support for zero-sized arrays mandatory.
    typedef _GLIBCXX_STD_C:: _array_traits<_Tp, _Nm> _AT_Type;
    typename _AT_Type:: Type _M_elems;

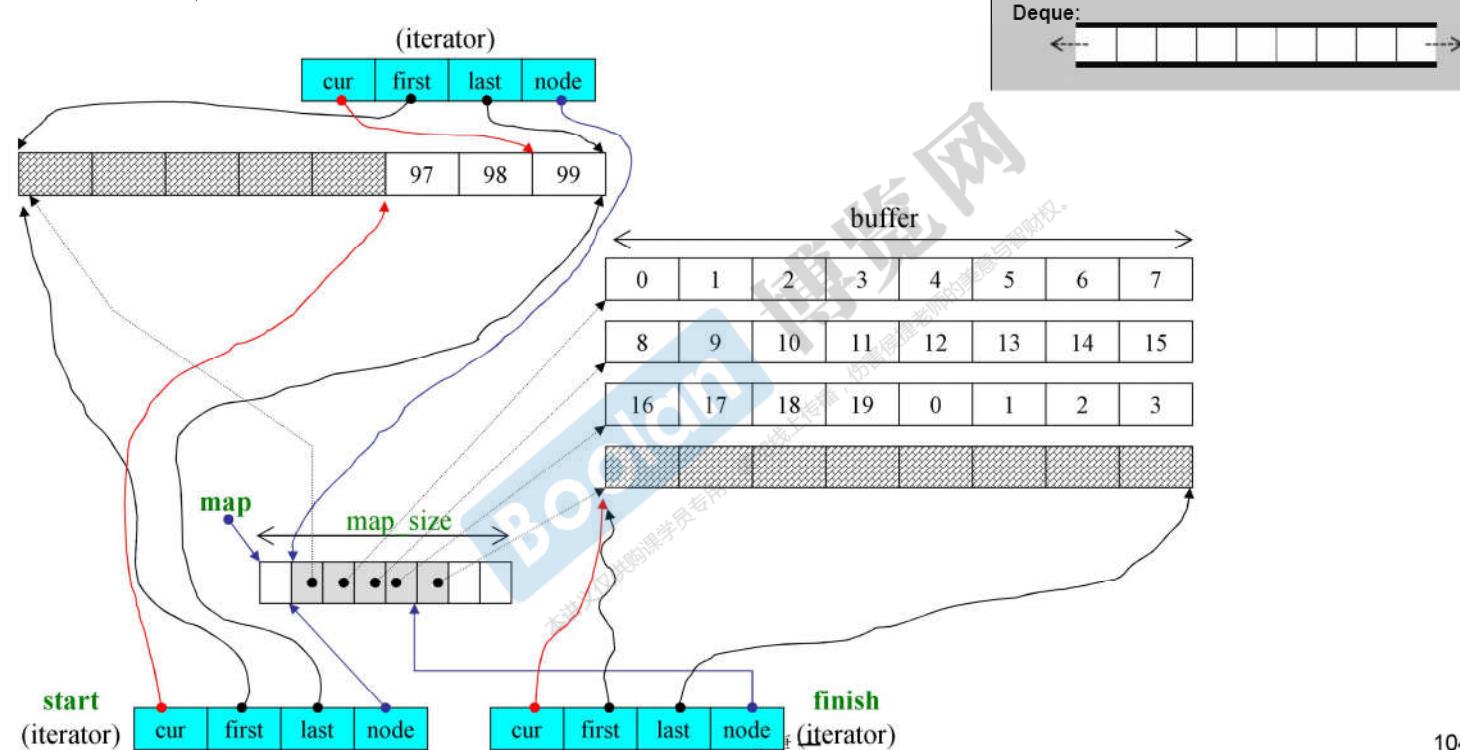
    // No explicit construct/copy/destroy for aggregate type.
    iterator begin() noexcept { return iterator(data()); }
    iterator end() noexcept { return iterator(data() + _Nm); }
    constexpr size_type size() const noexcept { return _Nm; }
    reference operator[](size_type __n) noexcept {
        return _AT_Type::_S_ref(_M_elems, __n); } ----- 沒有邊檢
    reference at(size_type __n) {
        if (__n >= _Nm) std::__throw_out_of_range_fmt(...); ----- 有邊檢
        return _AT_Type::_S_ref(_M_elems, __n);
    }
    pointer data() noexcept
    { return std::__addressof(_AT_Type::_S_ref(_M_elems, 0)); }
    ...
};

```

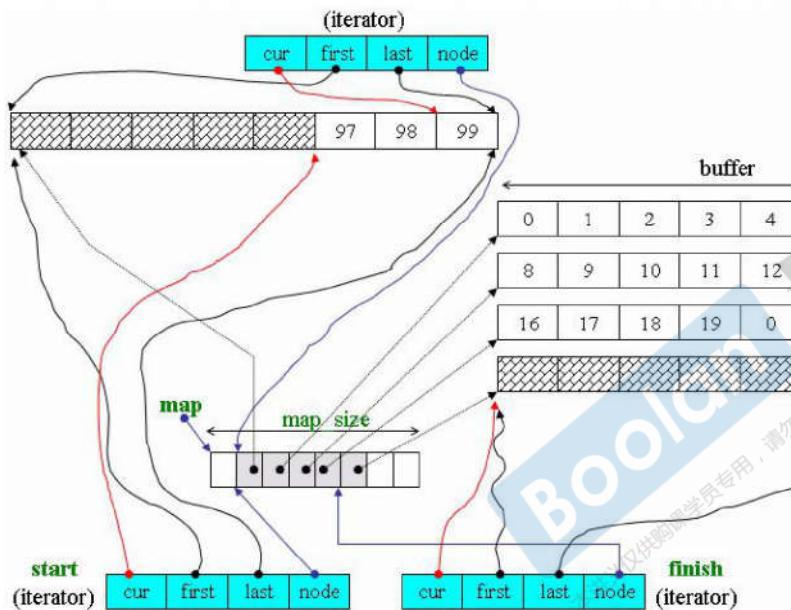
容器 forward_list



容器 deque



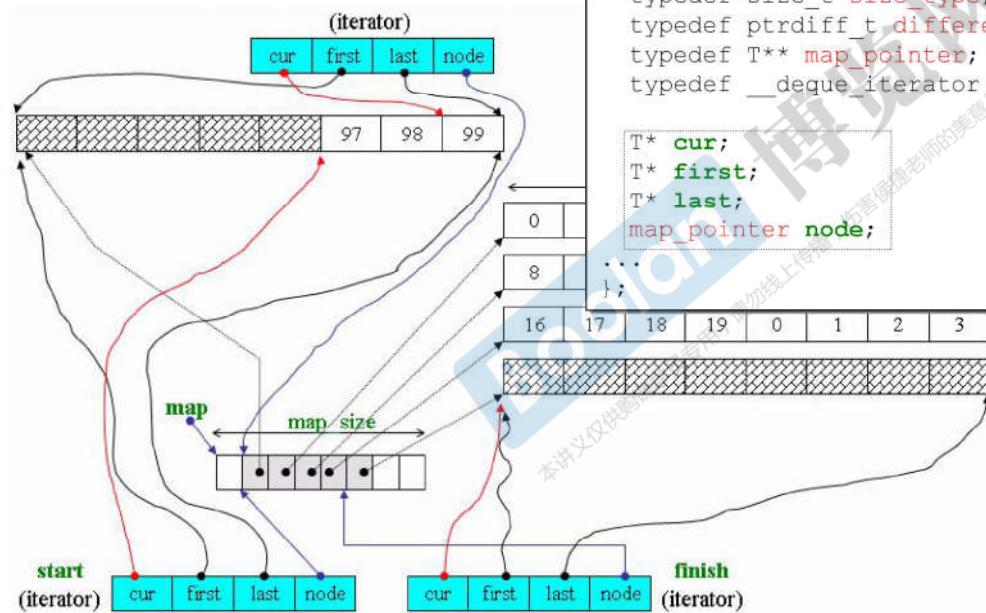
容器 deque



```
// 如果 n 不為 0，傳回 n，表示 buffer size 由使用者自定。  
// 如果 n 為 0，表示 buffer size 使用預設值，那麼  
//   如果 sz (sizeof(value_type)) 小於 512，傳回 512/sz，  
//   如果 sz 不小於 512，傳回 1。  
inline size_t __deque_buf_size(size_t n, size_t sz)  
{  
    return n != 0 ? n : (sz < 512 ? size_t(512 / sz) : size_t(1));  
}
```

```
template <class T, class Alloc=alloc,  
         size_t BufSiz=0>  
class deque {  
public:  
    // 所謂 buffer size 是指每個 buffer 容納的元素個數  
protected:  
    typedef T value_type;  
    typedef __deque_iterator<T, T&, T*, BufSiz> iterator;  
protected:  
    pointer* map_pointer; // T**  
protected:  
    iterator start;  
    iterator finish;  
    map_pointer map;  
    size_type map_size;  
public:  
    iterator begin() { return start; }  
    iterator end() { return finish; }  
    size_type size() const { return finish - start; }  
    ...  
};
```

///// deque'iterator

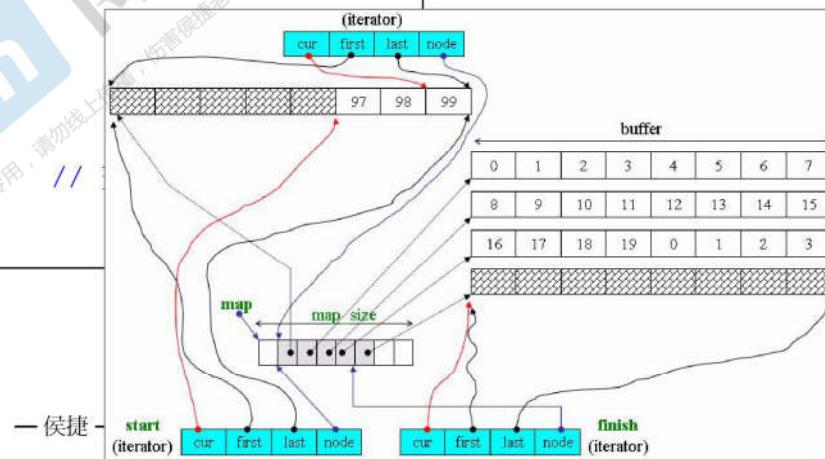


```
template <class T, class Ref, class Ptr, size_t BufSiz>
struct __deque_iterator {
    typedef random_access_iterator_tag iterator_category; // (1)
    typedef T value_type; // (2)
    typedef Ptr pointer; // (3)
    typedef Ref reference; // (4)
    typedef size_t size_type;
    typedef ptrdiff_t difference_type; // (5)
    typedef T** map_pointer;
    typedef __deque_iterator self;
};
```

deque<T>::insert()

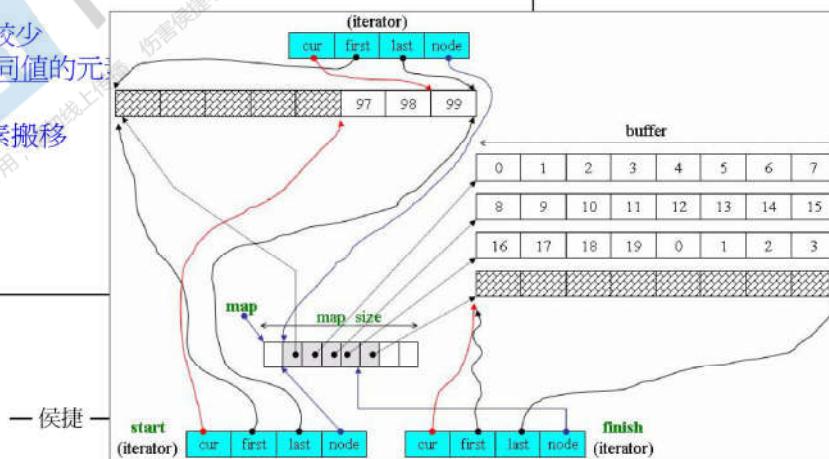
```
// 在position 處安插一個元素，其值為 x
iterator insert(iterator position, const value_type& x) {
    if (position.cur == start.cur) { //如果安插點是deque最前端
        push_front(x);           //交給push_front()做
        return start;
    }
    else if (position.cur == finish.cur) { //如果安插點是deque最尾端
        push_back(x);            //交給push_back()做
        iterator tmp = finish;
        --tmp;
        return tmp;
    }
    else {
        return insert_aux(position, x);
    }
}
```

下頁



■■■■ deque<T>::insert()

```
template <class T, class Alloc, size_t BufSize>
typename deque<T, Alloc, BufSize>::iterator
deque<T, Alloc, BufSize>::insert_aux(iterator pos, const value_type& x) {
    difference_type index = pos - start;           // 安插點之前的元素個數
    value_type x_copy = x;
    if (index < size() / 2) {                      // 如果安插點之前的元素個數較少
        push_front(front());                        // 在最前端加入與第一元素同值的元素。
        ...
        copy(front2, pos1, front1);                // 元素搬移
    }
    else {                                         // 安插點之後的元素個數較少
        push_back(back());                         // 在尾端加入與最末元素同值的元素。
        ...
        copy_backward(pos, back2, back1);          // 元素搬移
    }
    *pos = x_copy;      // 在安插點上設定新值
    return pos;
}
```



deque 如何模擬連續空間

全都是 deque iterators 的功勞

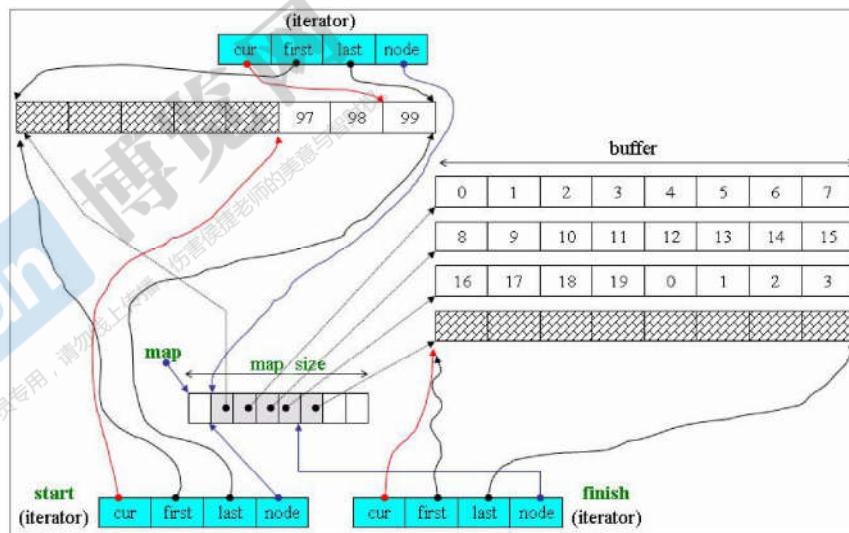
```
reference operator[](size_type n)
{
    return start[difference_type(n)];
}

reference front()
{ return *start; }

reference back()
{
    iterator tmp = finish;
    --tmp;
    return *tmp;
}

size_type size() const
{ return finish - start; }

bool empty() const
{ return finish == start; }
```



deque 如何模擬連續空間

全都是 deque iterators 的功勞

```
reference operator*() const
{ return *cur; }

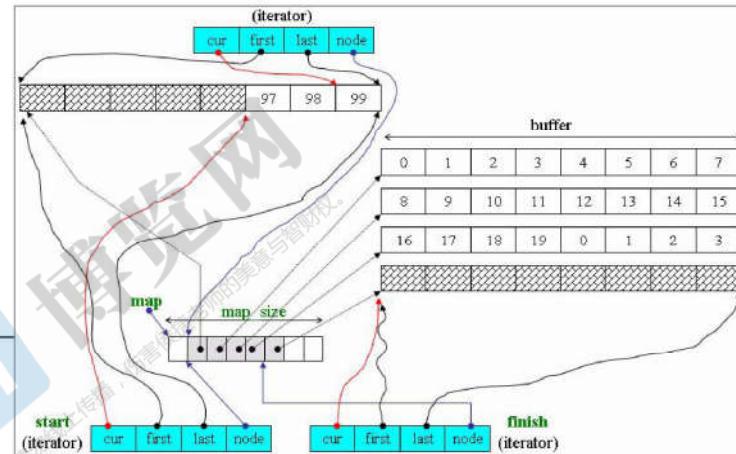
pointer operator->() const
{ return &(operator*()); }
```

```
//兩根iterators之間的距離相當於
//(1)兩根iterators間的buffers的總長度 +
//(2)itr至其buffer末尾的長度 +
//(3)x 至其buffer起頭的長度
difference_type
operator-(const self& x) const
{
    return difference_type(buffer_size()) * (node - x.node - 1) +
           (cur - first) + (x.last - x.cur);
}
```

末尾(當前) buffer
的元素量

起始 buffer
的元素量

首尾 buffers 之間的
buffers 數量



deque 如何模擬連續空間 全都是 deque iterators 的功勞

```
self& operator++() {
    ++cur; //切換至下一元素。
    if (cur == last) { //如果抵達緩衝區尾端,
        set_node(node + 1); //就跳至下一節點(緩衝區)
        cur = first; // 的起點。
    }
    return *this;
}
```

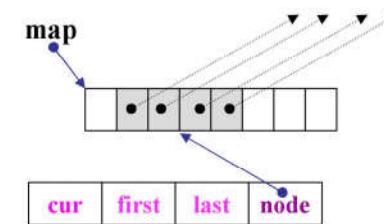
```
self operator++(int) {
    self tmp = *this;
    ++*this;
    return tmp;
}
```

```
self& operator--() {
    if (cur == first) { //如果目前在緩衝區起頭,
        set_node(node - 1); //就跳至前一節點(緩衝區)
        cur = last; // 的最末端。
    }
    --cur;
    return *this;
}
```

```
//往前移一元素(此即最末元素)
```

```
self operator--(int) {
    self tmp = *this;
    --*this;
    return tmp;
}
```

```
void set_node(map_pointer new_node) {
    node = new_node;
    first = *new_node;
    last = first +
           difference_type(buffer_size());
}
```





deque 如何模擬連續空間

全都是 deque iterators 的功勞

```
self& operator+=(difference_type n) {
    difference_type offset = n + (cur - first);
    if (offset >= 0 && offset < difference_type(buffer_size()))
        //目標位置在同一緩衝區內
        cur += n;
    else {
        //目標位置不在同一緩衝區內
        difference_type node_offset =
            offset > 0 ? offset / difference_type(buffer_size())
                         : -difference_type((-offset - 1) / buffer_size()) - 1;
        // 切換至正確的緩衝區
        set_node(node + node_offset);
        // 切換至正確的元素
        cur = first + (offset - node_offset * difference_type(buffer_size()));
    }
    return *this;
}

self operator+(difference_type n) const {
    self tmp = *this;
    return tmp += n;
}
```



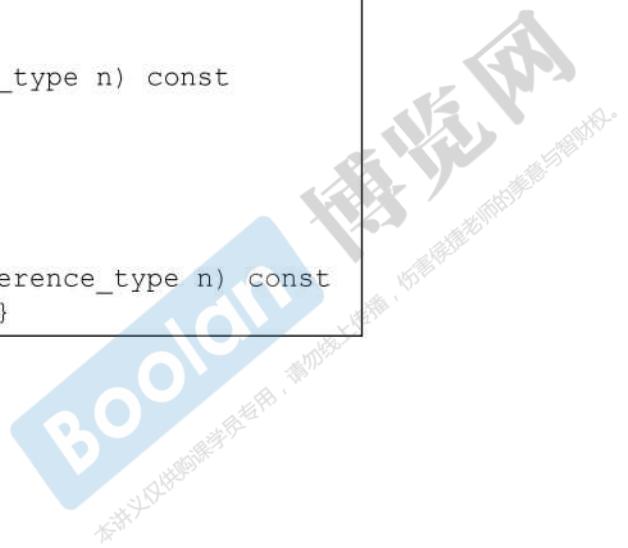
deque 如何模擬連續空間

全都是 deque iterators 的功勞

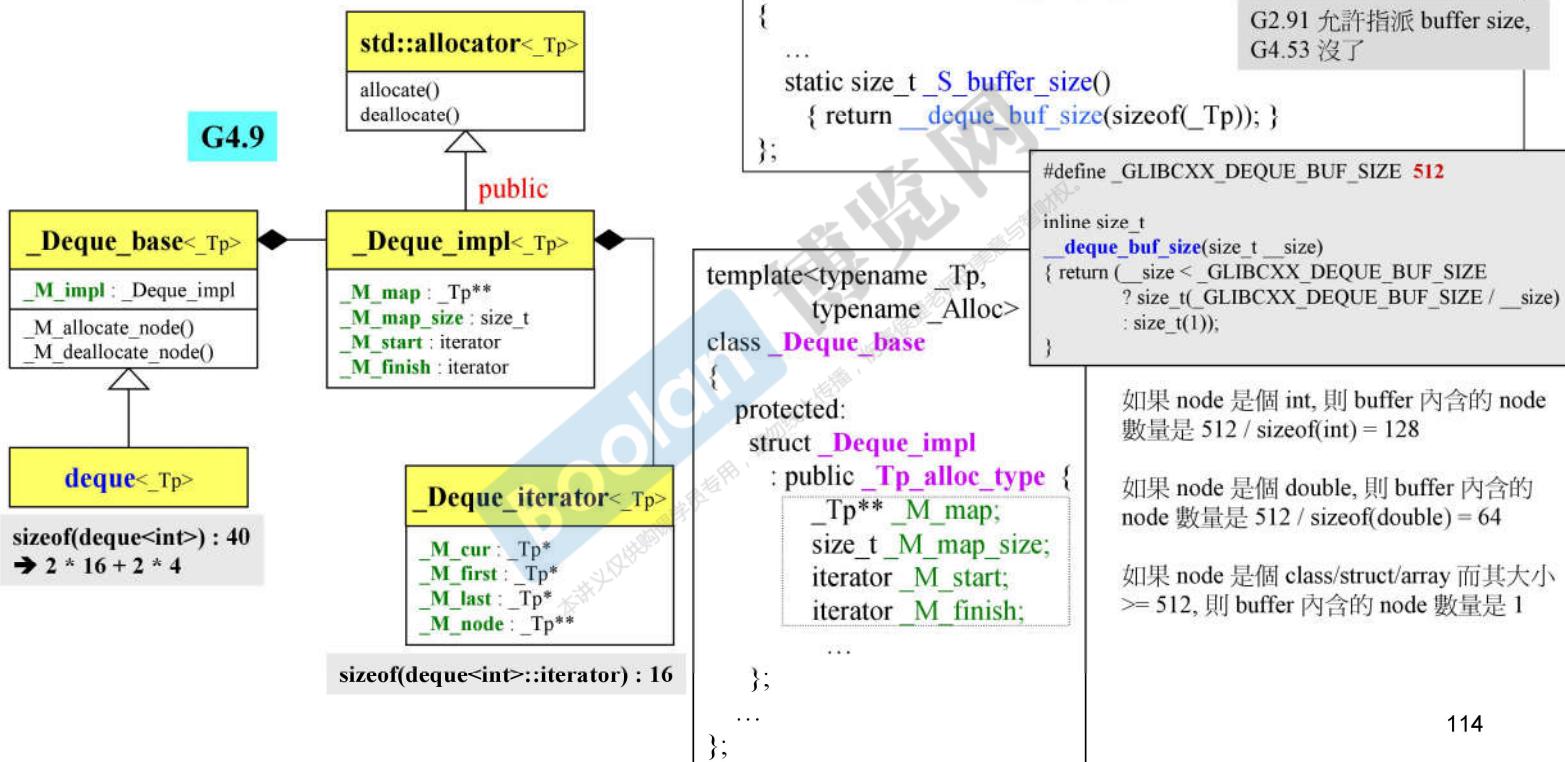
```
self& operator-=(difference_type n)
{ return *this += -n; }

self operator-(difference_type n) const
{
    self tmp = *this;
    return tmp -= n;
}

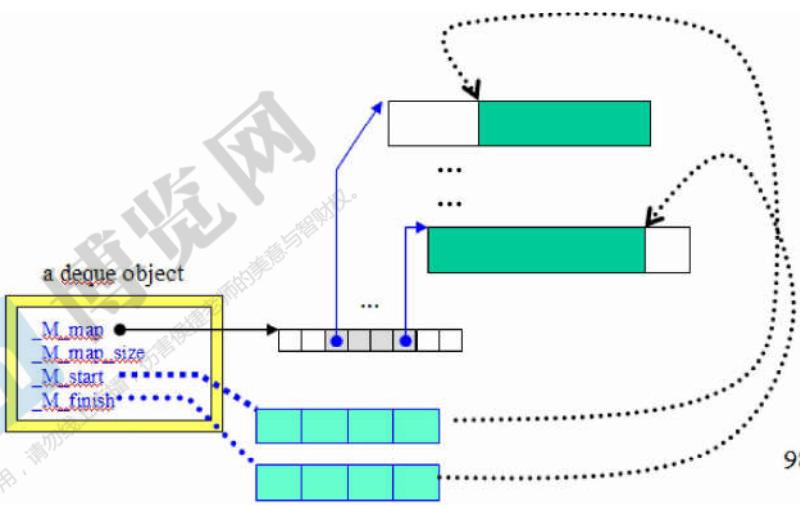
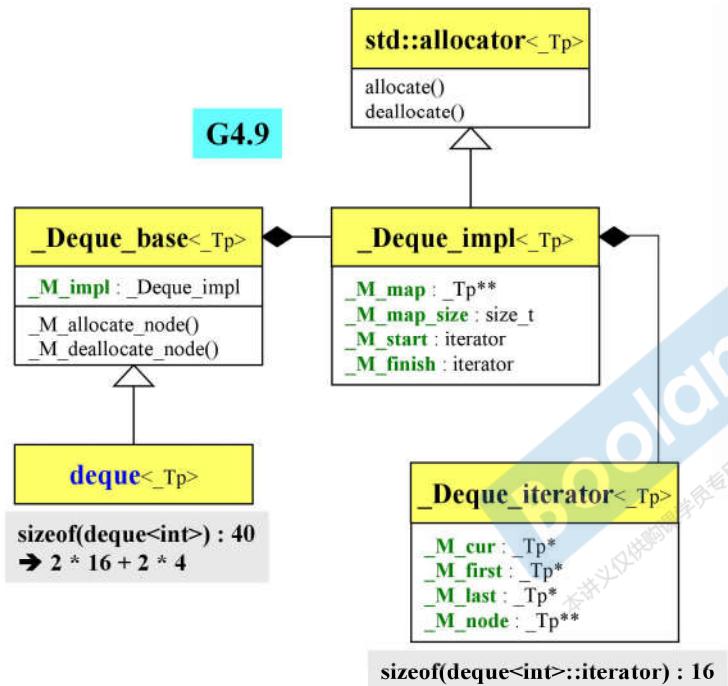
reference operator[](difference_type n) const
{ return *(*this + n); }
```



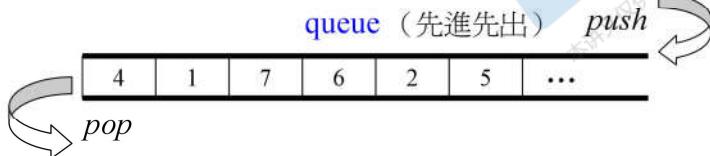
deque



///// deque

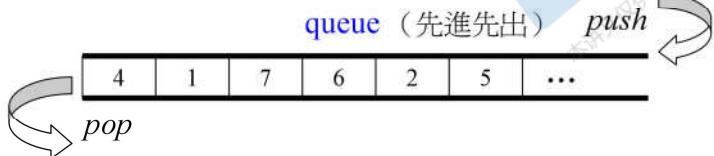


容器 queue



```
template <class T, class Sequence=deque<T>>
class queue {
...
public:
    typedef typename Sequence::value_type value_type;
    typedef typename Sequence::size_type size_type;
    typedef typename Sequence::reference reference;
    typedef typename Sequence::const_reference const_reference;
protected:
    Sequence c; //底層容器
public:
    bool empty() const { return c.empty(); }
    size_type size() const { return c.size(); }
    reference front() { return c.front(); }
    const_reference front() const { return c.front(); }
    reference back() { return c.back(); }
    const_reference back() const { return c.back(); }
    void push(const value_type& x) { c.push_back(x); }
    void pop() { c.pop_front(); }
};
```

容器 stack



```
template <class T, class Sequence=deque<T>>
class stack {
...
public:
    typedef typename Sequence::value_type value_type;
    typedef typename Sequence::size_type size_type;
    typedef typename Sequence::reference reference;
    typedef typename Sequence::const_reference const_reference;
protected:
    Sequence c;      // 底層容器
public:
    bool empty() const { return c.empty(); }
    size_type size() const { return c.size(); }
    reference top() { return c.back(); }
    const_reference top() const { return c.back(); }
    void push(const value_type& x) { c.push_back(x); }
    void pop() { c.pop_back(); }
};
```

queue 和 stack, 關於其 iterator 和底層結構

stack 或 queue 都不允許遍歷，
也不提供 iterator。

```
stack<string>::iterator ite; // [Error] 'iterator' is not a member of 'std::stack<std::basic_string<char>>'
```

```
queue<string>::iterator ite; // [Error] 'iterator' is not a member of 'std::queue<std::basic_string<char>>'
```

stack 和 queue 都可選擇 list 或 deque
做為底層結構。

```
stack<string, list<string>> c;
for(long i=0; i< 10; ++i) {
    sprintf(buf, 10, "%d", rand());
    c.push(string(buf));
}
cout << "stack.size()= " << c.size() << endl;
cout << "stack.top()= " << c.top() << endl;
c.pop();
cout << "stack.size()= " << c.size() << endl;
cout << "stack.top()= " << c.top() << endl;
```

```
queue<string, list<string>> c;
for(long i=0; i< 10; ++i) {
    sprintf(buf, 10, "%d", rand());
    c.push(string(buf));
}
cout << "queue.size()= " << c.size() << endl;
cout << "queue.front()= " << c.front() << endl;
cout << "queue.back()= " << c.back() << endl;
c.pop();
cout << "queue.size()= " << c.size() << endl;
cout << "queue.front()= " << c.front() << endl;
cout << "queue.back()= " << c.back() << endl;
```

一候

queue 和 stack, 關於其 iterator 和底層結構

queue 不可選擇 vector 做為底層結構

```
queue<string, vector<string>> c;
for(long i=0; i< 10; ++i) {
    sprintf(buf, 10, "%d", rand());
    c.push(string(buf));
}
cout << "queue.size()= " << c.size() << endl;
cout << "queue.front()= " << c.front() << endl;
cout << "queue.back()= " << c.back() << endl;
//c.pop(); //Error: 'class std::vector<std::basic_string<char> >' has no member named 'pop_front'
cout << "queue.size()= " << c.size() << endl;
cout << "queue.front()= " << c.front() << endl;
cout << "queue.back()= " << c.back() << endl;
```

stack 可選擇 vector 做為底層結構

```
stack<string, vector<string>> c;
for(long i=0; i< 10; ++i) {
    sprintf(buf, 10, "%d", rand());
    c.push(string(buf));
}
cout << "stack.size()= " << c.size() << endl;
cout << "stack.top()= " << c.top() << endl;
c.pop();
cout << "stack.size()= " << c.size() << endl;
cout << "stack.top()= " << c.top() << endl;
```

queue 和 stack, 關於其 iterator 和底層結構

stack 和 queue 都不可選擇 set 或 map 做為底層結構

```
stack<string, set<string>> c;
for(long i=0; i< 10; ++i) {
    snprintf(buf, 10, "%d", rand());
    c.push(string(buf));
}
cout << "stack.size()= " << c.size() << endl;
cout << "stack.top()= " << c.top() << endl;
c.pop();
cout << "stack.size()= " << c.size() << endl;
cout << "stack.top()= " << c.top() << endl;
```

編譯器 無異議



[Error] 'class std::set<std::basic_string<char> >' has no member named 'push_back'
[Error] 'class std::set<std::basic_string<char> >' has no member named 'back'
[Error] 'class std::set<std::basic_string<char> >' has no member named 'pop_back'

stack<string, map<string>> c;	[Error] template argument...
queue<string, map<string>> c;	[Error] template argument ...

容器 rb_tree

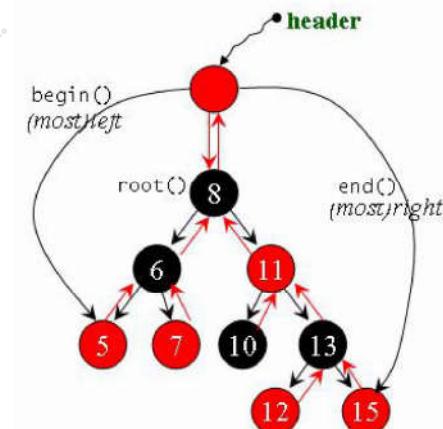
Red-Black tree（紅黑樹）是平衡二元搜尋樹（balanced binary search tree）中常被使用的一種。
平衡二元搜尋樹的特徵：排列規則有利 search 和 insert，並保持適度平衡—無任何節點過深。

rb_tree 提供“遍歷”操作及 iterators。

按正常規則 (`++ite`) 遍歷，便能獲得排序狀態（sorted）。

我們不應使用 rb_tree 的 iterators 改變元素值（因為元素有其嚴謹排列規則）。編程層面 (programming leve) 並未阻絕此事。如此設計是正確的，因為 rb_tree 即將為 set 和 map 服務（做為其底部支持），而 map 允許元素的 data 被改變，只有元素的 key 才是不可被改變的。

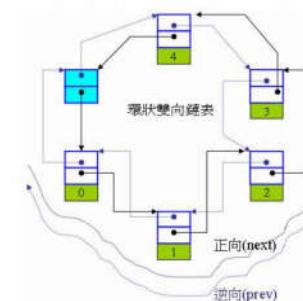
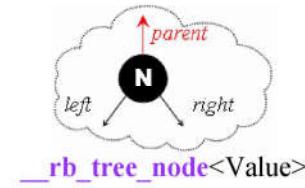
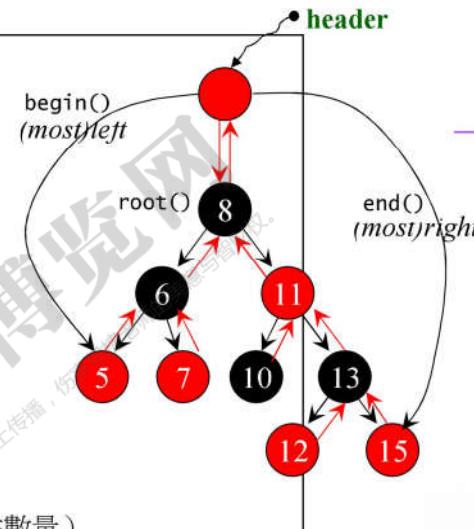
rb_tree 提供兩種 insertion 操作：`insert_unique()` 和 `insert_equal()`。
前者表示節點的 key 一定在整個 tree 中獨一無二，否則安插失敗；
後者表示節點的 key 可重複。



容器 rb_tree

```
template <class Key,
          class Value,
          class KeyOfValue,
          > class Compare,
          class Alloc = alloc>
class rb_tree {
protected:
    typedef __rb_tree_node<Value> rb_tree_node;
    ...
public:
    typedef rb_tree_node* link_type;
    ...
protected:
    // RB-tree 只以三筆資料表現它自己
    size_type node_count;           //rb_tree的大小(節點數量)
    link_type header;               //key的大小比較準則；應會是個 function object
    ...
};
```

大小 : 9 → 12



— 侯捷 —

容器 rb_tree

App.

```
rb_tree<int,  
        int,  
        identity<int>,  
        less<int>,  
        alloc>  
  
myTree;
```

C++ 標準庫

```
template <class Key,  
         class Value,  
         class KeyOfValue,  
         class Compare,  
         class Alloc = alloc>  
  
class rb_tree {  
    ...  
};
```

key | data

key 和 data
合成 value

```
template <class Arg, class Result>  
struct unary_function {  
    typedef Arg argument_type;  
    typedef Result result_type;  
};
```

```
template <class Arg1, class Arg2, class Result>  
struct binary_function {  
    typedef Arg1 first_argument_type;  
    typedef Arg2 second_argument_type;  
    typedef Result result_type;  
};
```

```
template <class T>  
struct identity : public unary_function<T, T> {  
    const T& operator()(const T& x) const { return x; }  
};
```

```
template <class T>  
struct less : public binary_function<T, T, bool> {  
    bool operator()(const T& x, const T& y) const  
    { return x < y; }  
};
```

容器 rb_tree, 用例

G2.9

```
//測試 rb_tree
rb_tree<int, int, identity<int>, less<int>> itree;
cout << itree.empty() << endl; //1
cout << itree.size() << endl; //0

itree.insert_unique(3);
itree.insert_unique(8);
itree.insert_unique(5);
itree.insert_unique(9);
itree.insert_unique(13);
itree.insert_unique(5); //no effect, since using insert_unique().
cout << itree.empty() << endl; //0
cout << itree.size() << endl; //5
cout << itree.count(5) << endl; //1

itree.insert_equal(5);
itree.insert_equal(5);
cout << itree.size() << endl; //7, since using insert_equal().
cout << itree.count(5) << endl; //3
```

容器 `_Rb_tree`, 用例

G4.9

```
//測試 rb_tree
_Rb_tree<int, int, _Identity<int>, less<int>> itree;
cout << itree.empty() << endl; //1
cout << itree.size() << endl; //0

itree._M_insert_unique(3);
itree._M_insert_unique(8);
itree._M_insert_unique(5);
itree._M_insert_unique(9);
itree._M_insert_unique(13);
itree._M_insert_unique(5); //no effect, since using _M_insert_unique().
cout << itree.empty() << endl; //0
cout << itree.size() << endl; //5
cout << itree.count(5) << endl; //1

itree._M_insert_equal(5);
itree._M_insert_equal(5);
cout << itree.size() << endl; //7, since using _M_insert_equal().
cout << itree.count(5) << endl; //3
```

容器 _Rb_tree, 用例

G4.9

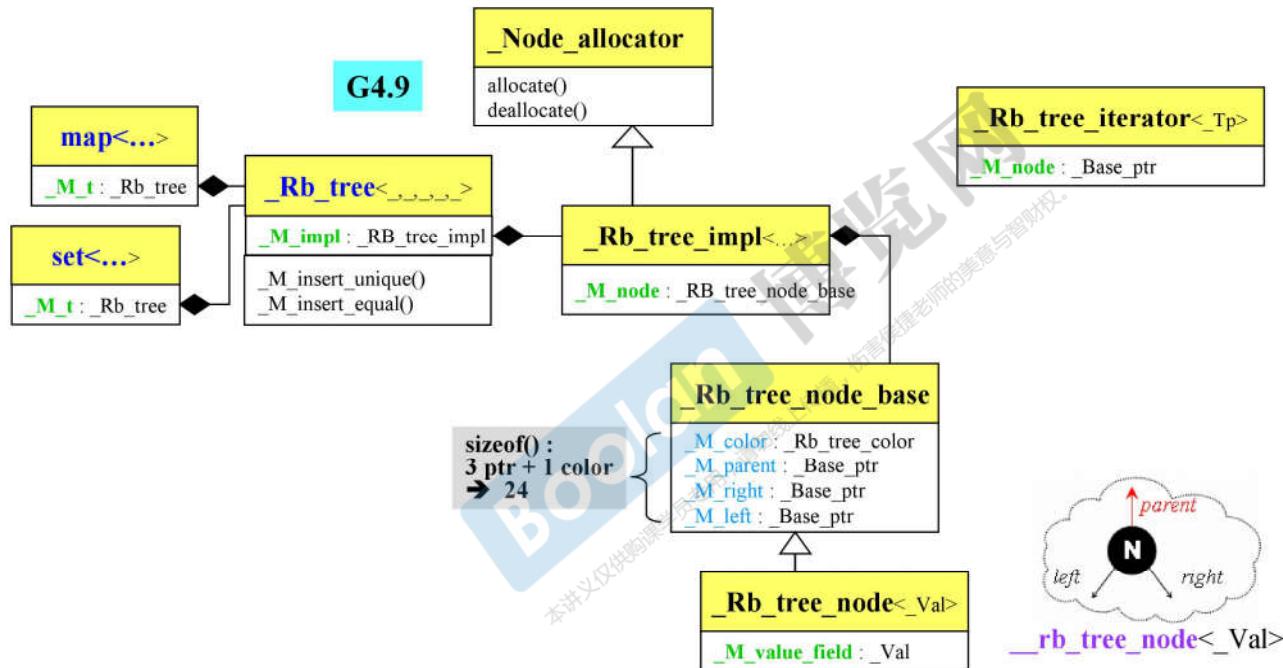
```
_Rb_tree<int, int, _Identity<int>, less<int>> itree;
cout << itree.empty() << endl; //1
cout << itree.size() << endl; //0

itree._M_insert_unique(3);
itree._M_insert_unique(8);
itree._M_insert_unique(5);
itree._M_insert_unique(9);
itree._M_insert_unique(13);
itree._M_insert_unique(5); //no effect, since using insert_unique().
cout << itree.empty() << endl; //0
cout << itree.size() << endl; //5
cout << itree.count(5) << endl; //1

itree._M_insert_equal(5);
itree._M_insert_equal(5);
cout << itree.size() << endl; //7, since using insert_equal().
cout << itree.count(5) << endl; //3
```

```
c:\ D:\handoutC++11-test... test_Rb_tree().....  
1  
0  
0  
5  
1  
7  
3
```

容器 _Rb_tree



— 侯捷 —

127

容器 set, multiset

set/multiset 以 rb_tree 為底層結構，因此有「元素自動排序」特性。
排序的依據是 key，而 set/multiset 元素的 value 和 key 合一：
value 就是 key。

key | data
key 和 data
合成 value

set/multiset 提供“遍歷”操作及 iterators。
按正常規則 (`++ite`) 遍歷，便能獲得排序狀態 (sorted)。

我們無法使用 set/multiset 的 iterators 改變元素值（因為 key 有其嚴謹排列規則）。set/multiset 的 iterator 是其底部的 RB tree 的 const-iterator，就是為了禁止 user 對元素賦值。

set 元素的 key 必須獨一無二，因此其 `insert()` 用的是 rb_tree 的 `insert_unique()`。
multiset 元素的 key 可以重複，因此其 `insert()` 用的是 rb_tree 的 `insert_equal()`。

容器 set

```
template <class Key,
          class Compare = less<Key>,
          class Alloc = alloc>
class set {
public:
    // typedefs:
    typedef Key key_type;
    typedef Key value_type;
    typedef Compare key_compare;
    typedef Compare value_compare;
private:
    typedef rb_tree<key_type, value_type,
                    identity<value_type>, key_compare, Alloc> rep_type;
    rep_type t;
public:
    typedef typename rep_type::const_iterator iterator;
...
};
```

key | data

key 和 data
合成 value

set<int> iset;

set<
int,
less<int>,
alloc
> iset;

template <
int,
int,
identity<int>,
less<int>,
alloc
>
class rb_tree;

set 的所有操作，都轉呼叫底層 t 的操作。
從這層意義看，set 未嘗不是個 container adapter。

容器 set, in VC6

VC6 不提供 identity()，那麼其 set 和 map 如何使用 RB-tree ?

```
template<class _K, class _Pr = less<_K>,
         class _A = allocator<_K>>
class set {
public:
    typedef set<_K, _Pr, _A> _Myt;
    typedef _K value_type;
    struct _Kfn : public unary_function<value_type, _K> {
        const _K& operator()(const value_type& _X) const
        {return (_X); }
    };
    typedef _Pr value_compare;
    typedef _K key_type;
    typedef _Pr key_compare;
    typedef _A allocator_type;
    typedef _Tree<_K, value_type, _Kfn, _Pr, _A> _Imp;
    ...
protected:
    _Imp _Tr;
};
```

VC6

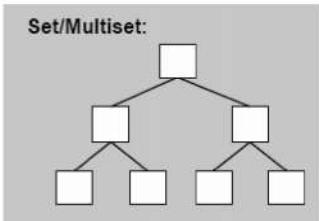
G2.9

```
template <class Arg, class Result>
struct unary_function {
    typedef Arg argument_type;
    typedef Result result_type;
};
```

```
template <class T>
struct identity : public unary_function<T, T> {
    const T& operator()(const T& x) const { return x; }
};
```



使用容器 multiset



```
D:\handout\C++11-test-DevC++\Test-STL>
select: 6
how many elements: 1000000

test_multiset<...>.....
milli-seconds : 6609
multiset.size()= 1000000
multiset.max_size()= 214748364
target <0^32767>: 23456
::find(), milli-seconds : 203
found, 23456
c.find(), milli-seconds : 0
found, 23456
```

```
313 void test_multiset(long& value)
314 {
315     cout << "\ntest_multiset()..... \n";
316     !
317     multiset<string> c;
318     char buf[10];
319     clock_t timeStart = clock();
320     for(long i=0; i< value; ++i)
321     {
322         try {
323             sprintf(buf, 10, "%d", rand());
324             c.insert(string(buf));
325         }
326         catch(exception& p) {
327             cout << "i=" << i << " " << p.what() << endl; // abort();
328         }
329     }
330     cout << "milli-seconds : " << (clock()-timeStart) << endl; //
331     cout << "multiset.size()=" << c.size() << endl;
332     cout << "multiset.max_size()=" << c.max_size() << endl;
333
334     string target = get_a_target_string();
335     {
336         timeStart = clock();
337         auto pItem = c.find(c.begin(), c.end(), target); //比 c.find(...) 慢很多
338         cout << "::find(), milli-seconds : " << (clock()-timeStart) << endl;
339         if (pItem != c.end())
340             cout << "found, " << *pItem << endl;
341         else
342             cout << "not found! " << endl;
343     }
344     {
345         timeStart = clock();
346         auto pItem = c.find(target); //比 ::find(...) 快很多
347         cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;
348         if (pItem != c.end())
349             cout << "found, " << *pItem << endl;
350         else
351             cout << "not found! " << endl;
352     }
353 }
```

容器 map, multimap

map/multimap 以 rb_tree 為底層結構，因此有「元素自動排序」特性。
排序的依據是 key。

key | data
key 和 data
合成 value

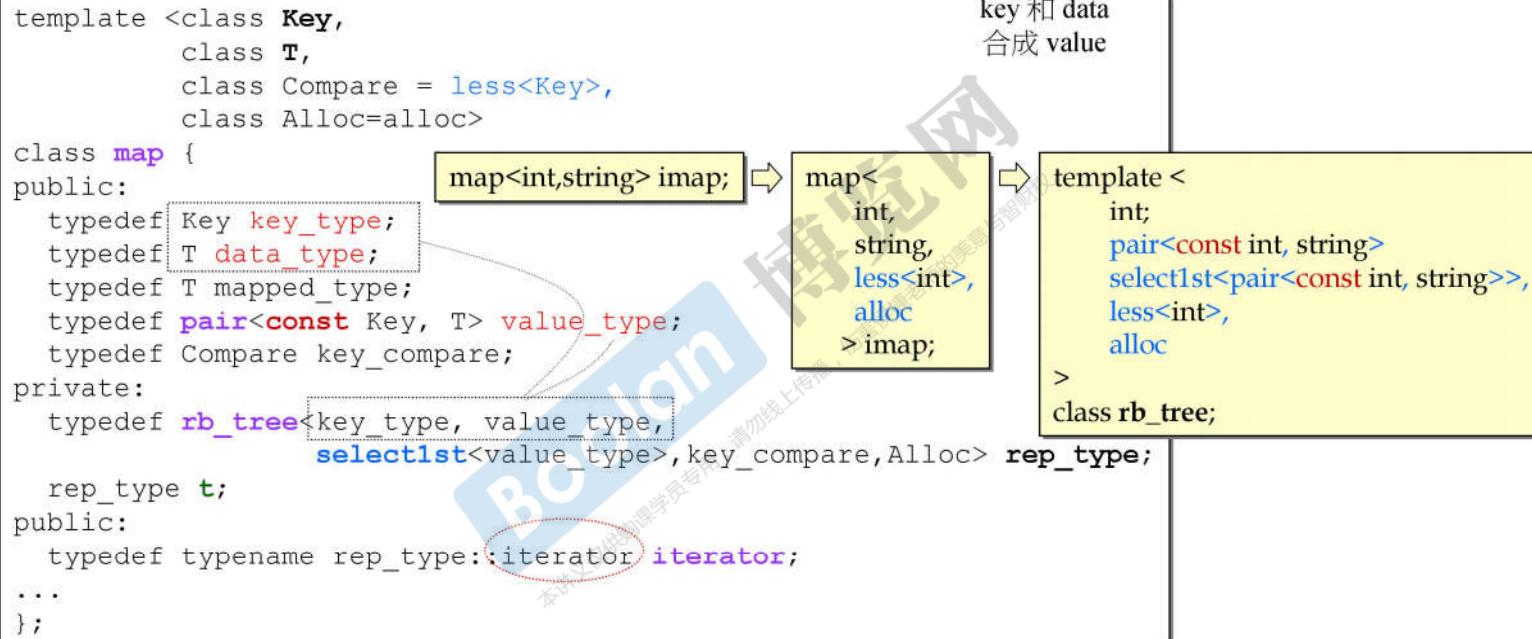
map/multimap 提供“遍歷”操作及 iterators。
按正常規則 (++ite) 遍歷，便能獲得排序狀態 (sorted)。

我們無法使用 map/multimap 的 iterators 改變元素的 key
(因為 key 有其嚴謹排列規則)，但可以用它來改變元素
的 data。因此 map/multimap 內部自動將 user 指定的 key
type 設為 **const**，如此便能禁止 user 對元素的 key 賦值。

```
template <class Key, class T, ...>
class map {
    typedef pair<const Key, T> value_type;
    typedef
        rb_tree<key_type,value_type, ...> rep_type;
    rep_type t;
    ...
}
```

map 元素的 key 必須獨一無二，因此其 insert() 用的是 rb_tree 的 insert_unique()。
multimap 元素的 key 可以重複，因此其 insert() 用的是 rb_tree 的 insert_equal()。

容器 map



容器 map, in VC6

VC6 不提供 select1st()，那麼其 map 如何使用 RB-tree ?

```
template<class _K, class _Ty, class _Pr = less<_K>, VC6  
        class _A = allocator<_Ty>>  
class map {  
public:  
    typedef map<_K, _Ty, _Pr, _A> _Myt;  
    typedef pair<const _K, _Ty> value_type;  
    struct _Kfn : public unary_function<value_type, _K> {  
        const _K& operator()(const value_type& _X) const  
        {return (_X.first); }  
    };  
    ...  
    typedef _Tree<_K, value_type, _Kfn, _Pr, _A> _Imp;  
    ...  
protected:  
    _Imp _Tr;  
};
```

G2.9

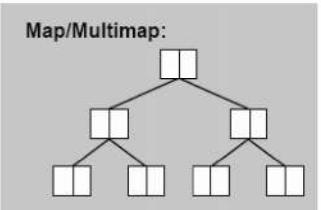
```
template <class Arg, class Result>  
struct unary_function {  
    typedef Arg argument_type;  
    typedef Result result_type;  
};
```

template <class Pair>

struct select1st :

```
public unary_function<Pair, typename Pair::first_type> {  
    const typename Pair::first_type&  
    operator()(const Pair& x) const  
    { return x.first; }  
};
```

使用容器 multimap



```
ex D:\handout\c++11-test-DevC++\Test-STL\m  
select: 7  
how many elements: 1000000  
  
test_multimap<>.  
milli-seconds : 4812  
multimap.size()= 1000000  
multimap.max_size()= 178956970  
target <0~32767>: 23456  
c.find(), milli-seconds : 0  
found, value=29247
```

```
364 namespace jj07  
365 {  
366     void test_multimap(long& value)  
367     {  
368         cout << "\ntest_multimap()..... \n";  
369  
370         multimap<long, string> c;  
371         char buf[10];  
372  
373         clock_t timeStart = clock();  
374         for(long i=0; i< value; ++i)  
375         {  
376             try {  
377                 snprintf(buf, 10, "%d", rand());  
378                 //multimap 不可使用 [] 做 insertion  
379                 c.insert(pair<long,string>(i,buf));  
380             }  
381             catch(exception& p) {  
382                 cout << "i=" << i << " " << p.what() << endl; //  
383                 abort();  
384             }  
385         }  
386         cout << "milli-seconds : " << (clock()-timeStart) << endl; //  
387         cout << "multimap.size()= " << c.size() << endl;  
388         cout << "multimap.max_size()= " << c.max_size() << endl;  
389  
390         long target = get_a_target_long();  
391         timeStart = clock();  
392         auto pItem = c.find(target);  
393         cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;  
394         if (pItem != c.end())  
395             cout << "found, value=" << (*pItem).second << endl;  
396         else  
397             cout << "not found! " << endl;  
398     }  
399 }
```

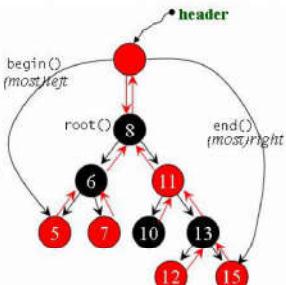
容器 map, 獨特的 operator[]

G4.9

```

mapped_type&
operator[](const key_type& __k)
{
    // concept requirements
    __glbcxx_function_requires(_DefaultConstructibleCon
    iterator __i = lower_bound(__k);
    // __i->first is greater than or equivalent to __k.
    if (__i == end() || key_comp)(__k, (*__i).first))
#if __cplusplus >= 201103L
        __i = _M_t._M_emplace_hint_unique(__i, std::piecewise_construct
                                            std::tuple<const key_type>
                                            std::tuple<>());
#else
        __i = insert(__i, value_type(__k, mapped_type()));
#endif
    return (*__i).second;
}

```



// [23.3.1.2] element access

```

/**
 * @brief Subscript (@c []) access to %map data.
 * @param __k The key for which data should be retrieved.
 * @return A reference to the data of the (key,data) %pair.
 *
 * Allows for easy lookup with the subscript (@c [])
 * operator. Returns data associated with the key specified in
 * subscript. If the key does not exist, a pair with that key
 * is created using default values, which is then returned.
 *
 * Lookup requires logarithmic time.
 */

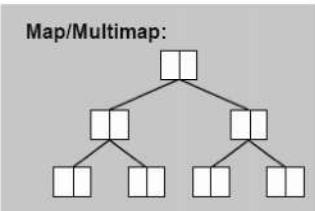
```

- `lower_bound` 是二分搜尋 (binary search) 的一種版本，試圖在 sorted $[first, last]$ 中尋找元素 `value`。若 $[first, last]$ 擁有與 `value` 相等的元素(s)，便返回一個 `iterator` 指向其中第一個元素。如果沒有這樣的元素存在，便返回「假設該元素存在時應該出現的位置」。也就是說它會返回 `iterator` 指向第一個「不小於 `value`」的元素。如果 `value` 大於 $[first, last]$ 內的任何元素，將返回 `last`。換句話說 `lower_bound` 返回的是「不破壞排序得以安插 `value` 的第一個適當位置」。

— 侯捷 —



使用容器 map



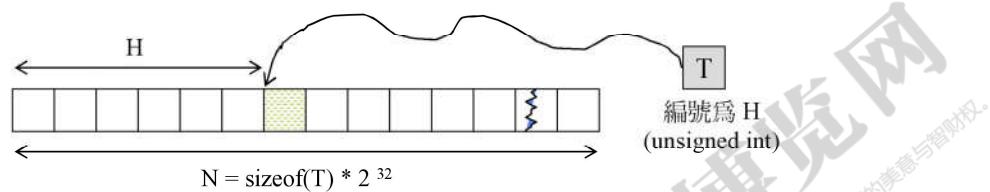
```
ex D:\handout\c++11-test-DevC++\Test-STI
select: 14
how many elements: 1000000

test_map().....
milli-seconds : 4890
map.size()= 1000000
map.max_size()= 178956970
target <0~32767>: 23456
c.find(), milli-seconds : 0
found, value=19128
```

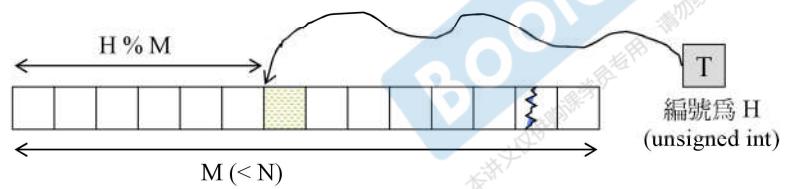
```
696 namespace jj14
697 {
698     void test_map(long& value)
699     {
700         cout << "\ntest_map()..... \n";
701
702         map<long, string> c;
703         char buf[10];
704
705         clock_t timeStart = clock();
706         for(long i=0; i< value; ++i)
707         {
708             try
709             {
710                 sprintf(buf, 10, "%d", rand());
711                 c[i] = string(buf);
712             }
713             catch(exception& p)
714             {
715                 cout << "i=" << i << " " << p.what() << endl;    //
716                 abort();
717             }
718         }
719         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
720         cout << "map.size()= " << c.size() << endl;
721         cout << "map.max_size()= " << c.max_size() << endl;
722
723         long target = get_a_target_long();
724         timeStart = clock();
725         auto pItem = c.find(target);
726         cout << "c.find(), milli-seconds : " << (clock()-timeStart) << endl;
727         if (pItem != c.end())
728             cout << "found, value=" << (*pItem).second << endl;
729         else
730             cout << "not found! " << endl;
731     }
732 }
```

容器 hashtable

空間足夠時：

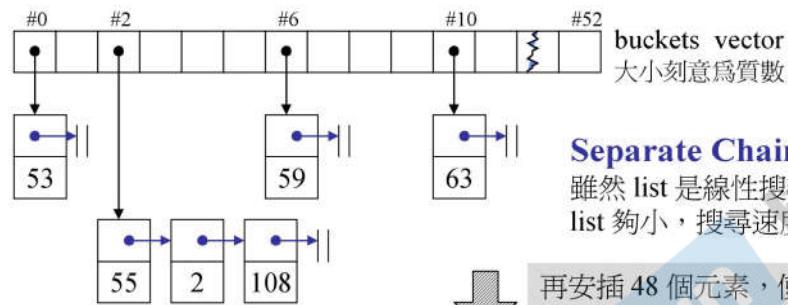


空間不足時：



容器 hashtable

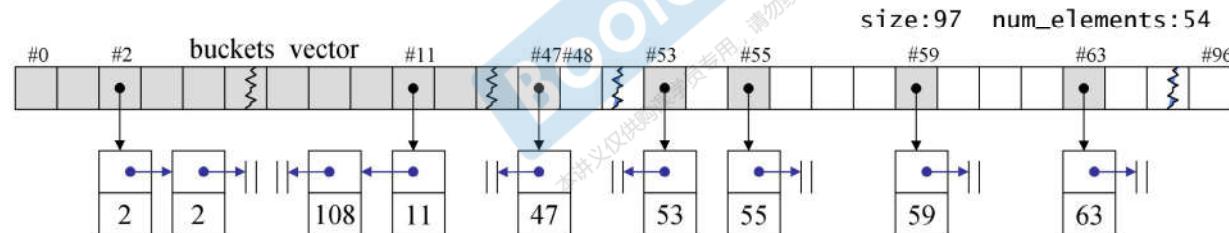
`insert_unique()`, 次序: 59, 63, 108, 2, 53, 55 num_elements: 6



Separate Chaining

雖然 list 是線性搜尋時間，如果 list 夠小，搜尋速度仍然很快

再安插 48 個元素，使總量達到 54 個，超過當時的 buckets vector 大小 53，於是 rehashing



我們可以使用 hashtable iterators 改變元素的 data，但不能改變元素的 key
(因為 hashtable 根據 key 實現嚴謹的元素排列)。

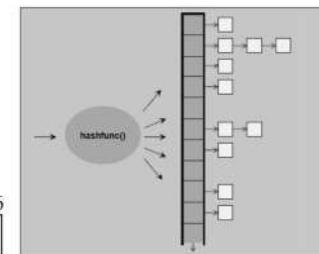
`static const unsigned long`

`_stl_prime_list[_stl_num_primes] =`

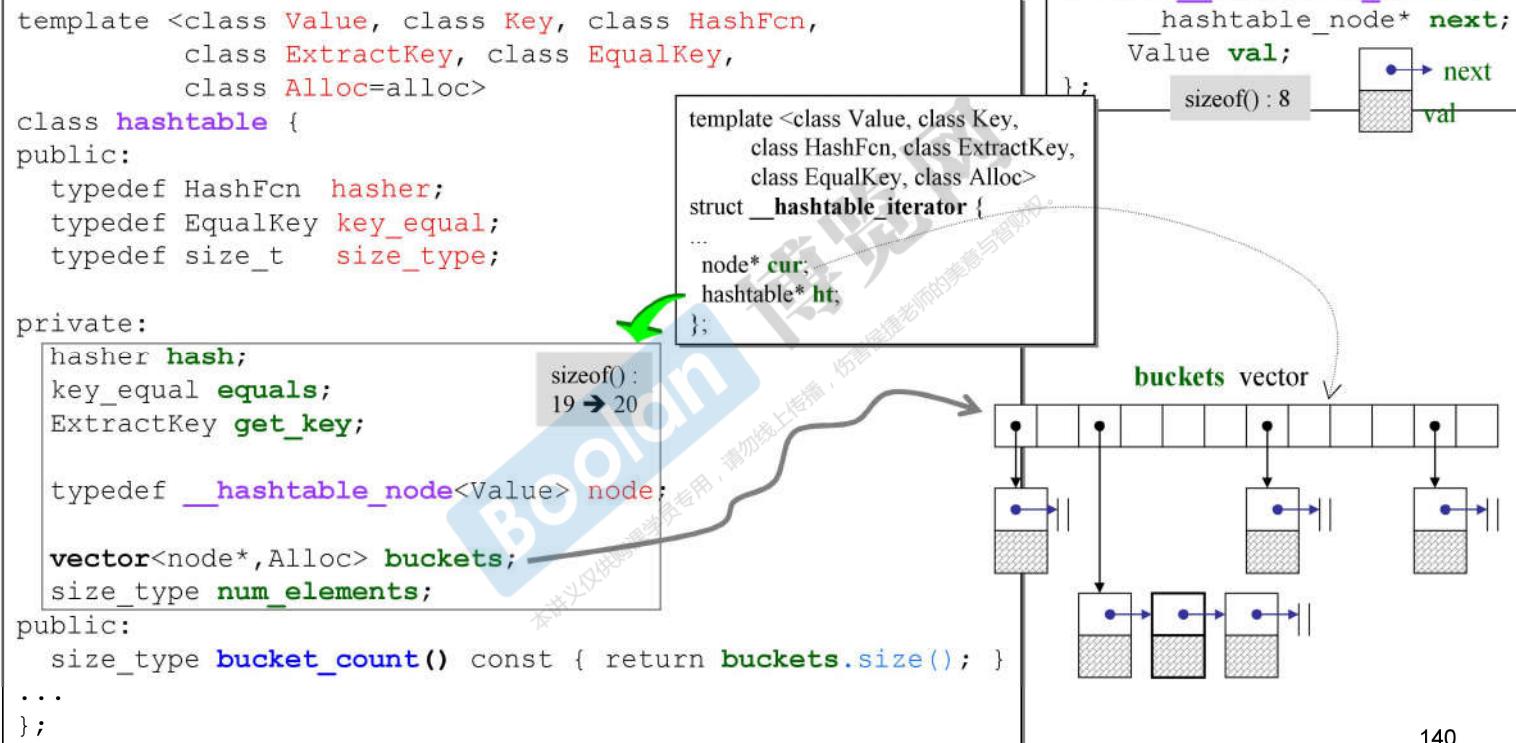
```
{
  53, 97, 193, 389, 769,
  1543, 3079, 6151, 12289, 24593,
  49157, 98317, 196613, 393241, 786433,
  1572869, 3145739, 6291469, 12582917, 25165843,
  50331653, 100663319, 201326611, 402653189,
  805306457, 1610612741, 3221225473ul, 4294967291ul
};
```

G2.9

x G4.9



容器 hashtable



容器 hashtable

App.

```
hashtable<const char*,  
          const char*,  
          hash<const char*>,  
          identity<const char*>,  
          eqstr,  
          alloc>  
  
ht(50,hash<const char*>(),eqstr());  
  
ht.insert_unique("kiwi");  
ht.insert_unique("plum");  
ht.insert_unique("apple");
```

比較兩個 c-string 是否相等，有 strcmp() 可用，但它傳回 -1,0,1，不是傳回 bool，所以必須加一層外套

```
struct eqstr {  
    bool operator()(const char* s1,  
                    const char* s2) const  
    { return strcmp(s1,s2) == 0; }  
};
```

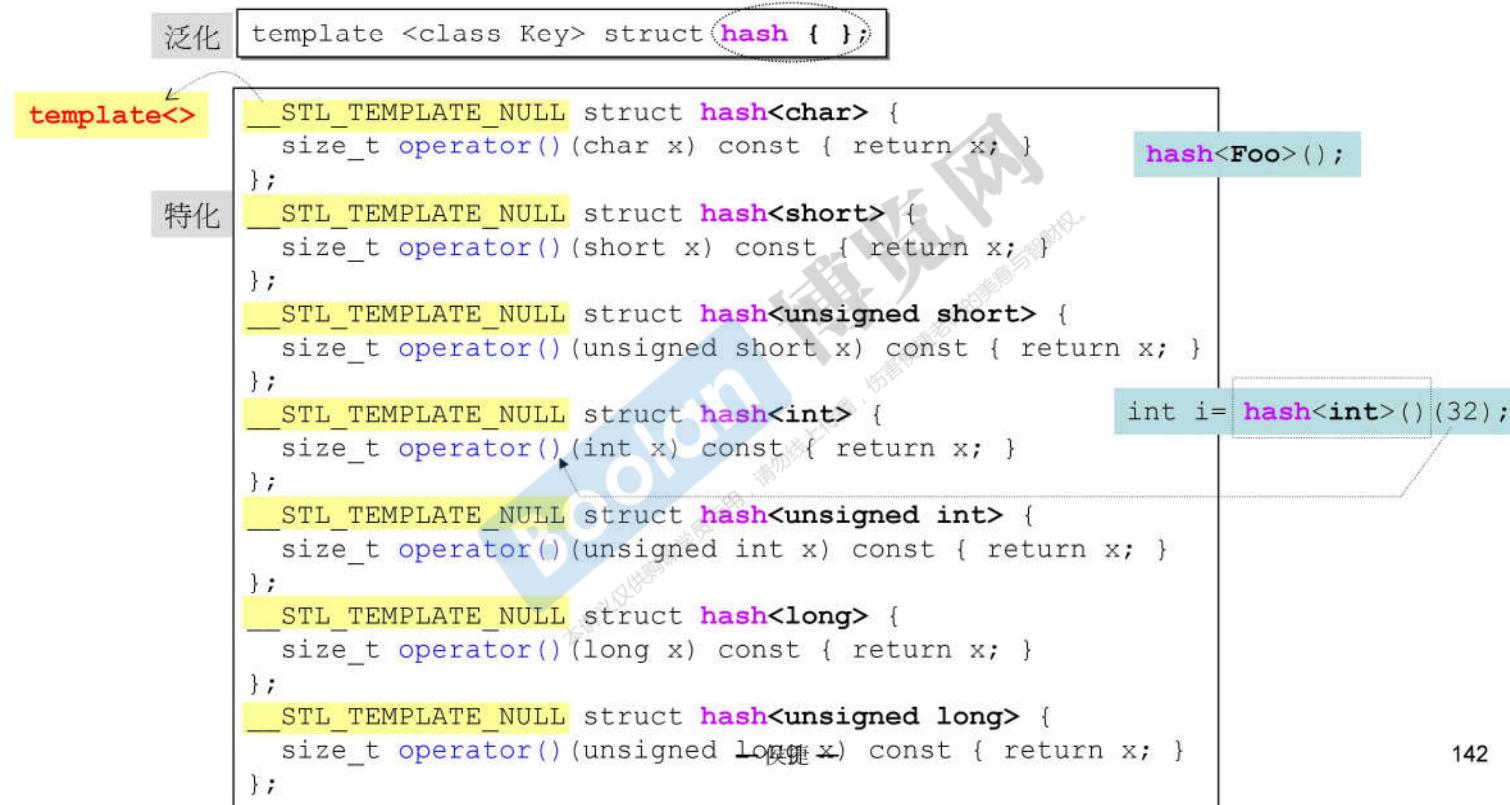
C++ 標準庫

```
template <class Value,  
         class Key,  
         class HashFcn,  
         class ExtractKey,  
         class EqualKey,  
         class Alloc=alloc>  
class hashtable {  
public:  
    typedef HashFcn    hasher;  
    typedef EqualKey   key_equal;  
    typedef size_t     size_type;  
private:  
    hasher    hash;  
    key_equal equals;  
    ExtractKey get_key;  
    ...
```

下頁

— 侯捷 —

hash-function, hash-code



hash-function, hash-code

```
inline size_t __stl_hash_string(const char* s)
{
    unsigned long h = 0;
    for ( ; *s; ++s) {
        h = 5*h + *s;           // 本例若 s 指向 "abc"，計算得 h 為
                                // 5 * (5 * ('a' + 'b') + 'c')
    }
    return size_t(h);
}

__STL_TEMPLATE_NULL struct hash<char*>
{
    size_t operator()(const char* s) const { return __stl_hash_string(s); }
};

__STL_TEMPLATE_NULL struct hash<const char*>
{
    size_t operator()(const char* s) const { return __stl_hash_string(s); }
};
```

注意，標準庫 **G2.9** 沒有提供現成的
`hash<std::string>`
G4.9 有提供，在
...\\4.9.2\\include\\c++\\bits\\basic_string.h

hash function 的目的，就是希望根據元素值算出一個 hash code（一個可進行 modulus 運算的值），使得元素經 hash code 映射之後能夠「夠雜夠亂夠隨機」地被置於 hashtable 內。愈是雜亂，愈不容易發生碰撞。

modulus 運算

```
iterator find(const key_type& key)
{ size_type n = bkt_num_key(key); //花落何家
...
size_type count(const key_type& key) const
{ const size_type n = bkt_num_key(key);
...
template <class V, class K, class HF, class ExK,
          class EqK, class A>
__hashtable_iterator<V, K, HF, ExK, EqK, A>&
__hashtable_iterator<V, K, HF, ExK, EqK, A>::operator++()
{
...
    size_type bucket = ht->bkt_num(old->val);
...
}
```

1. `bkt_num_key` (const key_type& key) const

```
{  
    return bkt_num_key(key, buckets.size());  
}  
2. bkt_num (const value_type& obj) const
```

size_type
`bkt_num_key` (const key_type& key, size_t n) const

```
{  
    return hash(key) % n;  
}
```

這個 `hash` 不是前頁出現的 struct `hash`，而是 class hashtable 中的 hasher `hash`。

size_type
`bkt_num` (const value_type& obj, size_t n) const

```
{  
    return bkt_num_key(get_key(obj), n);  
}
```

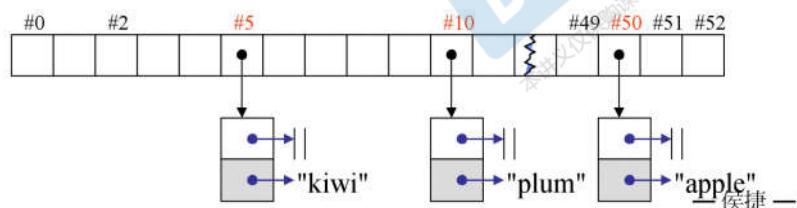
template <...>
class hashtable {
private:
 hasher hash;
...
}

— 侯捷 —

hash-function, hash-code

App.

```
hashtable<const char*,
           const char*,
           hash<const char*>,
           identity<const char*>,
           eqstr,
           alloc>
ht(50,hash<const char*>(),eqstr());
ht.insert_unique("kiwi");
ht.insert_unique("plum");
ht.insert_unique("apple");
```



$$\begin{aligned}\text{hash code of "kiwi"} &= 5 * (5 * (5 * 'k' + 'i') + 'w') + 'i' \\ &= 5 * (5 * (5 * 107 + 105) + 119) + 105 \\ &= 16700 \% 53 = 5\end{aligned}$$

$$\begin{aligned}\text{hash code of "plum"} &= 5 * (5 * (5 * 'p' + 'l') + 'u') + 'm' \\ &= 5 * (5 * (5 * 112 + 108) + 117) + 109 \\ &= 17394 \% 53 = 10\end{aligned}$$

$$\begin{aligned}\text{hash code of "apple"} &= 5 * (5 * (5 * (5 * 'a' + 'p') + 'p') + 'l') + 'e' \\ &= 5 * (5 * (5 * (5 * 97 + 112) + 112) + 108) + 101 \\ &= 78066 \% 53 = 50\end{aligned}$$

↑

```
inline size_t
__stl_hash_string(const char* s)
{
    unsigned long h = 0;
    for ( ; *s; ++s)
        h = 5*h + *s;
    return size_t(h);
}
```

///// hashtable 用例 G2.9

```
7 template<> struct hash<string>
8 {
9     size_t operator()(string s) const {
10         return __stl_hash_string(s.c_str());
11     }
12 };
```

```
50 hashtable< string,
51             string,
52             hash<string>,
53             identity<string>,
54             equal_to<string>,
55             alloc
56         >
57         sht(50, hash<string>(), equal_to<string>());
58
59 cout<< sht.size() << endl;           // 0
60 cout<< sht.bucket_count() << endl;    // 53
61
62 // my goal!
63 hashtable< pair<const string,int>,
64             string,
65             hash<string>,
66             selectist< pair<const string,int> >,
67             equal_to<string>,
68             alloc
69         >
70         siht(100, hash<string>(), equal_to<string>());
71
72 cout<< siht.size() << endl;           // 0
73 cout<< siht.bucket_count() << endl;    // 193
74
75 siht.insert_unique( make_pair(string("jjhou"),95) );
76 siht.insert_unique( make_pair(string("sabrina"),90) );
77 siht.insert_unique( make_pair(string("mjchen"),85) );
78 cout<< siht.size() << endl;           // 3
79 cout<< siht.bucket_count() << endl;    // 193
80 cout << siht.find(string("sabrina"))->second << endl; //90
81 cout << siht.find(string("jjhou"))->second << endl;   //95
82 cout << siht.find(string("mjchen"))->second << endl; //85
```

hashtable 用例 G4.9

```
1355     _Hashtable< string,
1356             string,
1357             hash<string>,
1358             _Identity<string>,
1359             equal_to<string>,
1360             > // [Error] wrong number of template arguments (6, should be 10)
1361     sht(50, hash<string>(), equal_to<string>());
1362
1363     cout<< sht.size() << endl;           // 0
1364     cout<< sht.bucket_count() << endl;    // 53
1365
1366 // my goal!
1367     _Hashtable< pair<const string,int>,
1368             string,
1369             hash<string>,
1370             _Selectlist< pair<const string,int> >,
1371             equal_to<string>,
1372             > // [Error] wrong number of template arguments (6, should be 10)
1373     siht(100, hash<string>(), equal_to<string>());
1374
1375     cout<< siht.size() << endl;           // 0
1376     cout<< siht.bucket_count() << endl;    // 193
1377
1378     siht.insert_unique( make_pair(string("jjhou"),95) );
1379     siht.insert_unique( make_pair(string("sabrina"),90) );
1380     siht.insert_unique( make_pair(string("mjchen"),85) );
1381     cout<< siht.size() << endl;           // 3
1382     cout<< siht.bucket_count() << endl;    // 193
1383     cout << siht.find(string("sabrina"))->second << endl; //90
1384     cout << siht.find(string("jjhou"))->second << endl;   //95
1385     cout << siht.find(string("mjchen"))->second << endl; //85
```

```
/// std::hash specialization for string.
template<>
...\\4.9.2\\include\\c++\\bits\\basic_string.h
► struct hash<string>
: public __hash_base<size_t, string>
{
size_t
operator()(const string& __s) const noexcept
{ return std::__Hash_impl::hash(__s.data(), __s.length()); }
};

template<>
struct __is_fast_hash<hash<string>> : std::false_type
{ };
```



容器 **hash_set, hash_multiset**

容器 **hash_map, hash_multimap**



||||| **unordered 容器**

Before C++11

- **hash_set**
- **hash_multiset**
- **hash_map**
- **hash_multimap**



Since C++11

- **unordered_set**
- **unordered_multiset**
- **unordered_map**
- **unordered_multimap**

G4.9

```
template <typename T,
         typename Hash = hash<T>,
         typename EqPred = equal_to<T>,
         typename Allocator = allocator<T> >
class unordered_set;
```

```
template <typename T,
         typename Hash = hash<T>,
         typename EqPred = equal_to<T>,
         typename Allocator = allocator<T> >
class unordered_multiset;
```

```
template <typename Key, typename T,
         typename Hash = hash<T>,
         typename EqPred = equal_to<T>,
         typename Allocator = allocator<pair<const Key, T> > >
class unordered_map;
```

```
template <typename Key, typename T,
         typename Hash = hash<T>,
         typename EqPred = equal_to<T>,
         typename Allocator = allocator<pair<const Key, T> > >
class unordered_multimap;
```



容器 `unordered_set`, `unordered_multiset`

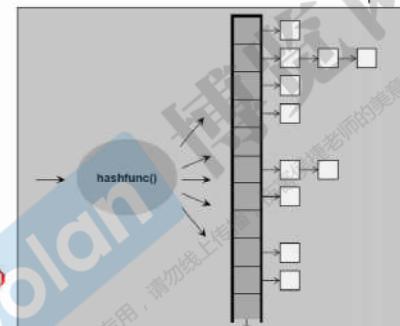
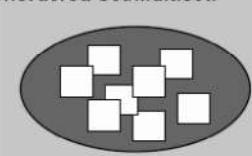
容器 `unordered_map`, `unordered_multimap`



使用容器 `unordered_set`

```
739 namespace jj15
740 {
741     void test_unordered_set(long& value)
742     {
743         cout << "\ntest_unordered_set().....\n";
744
745         unordered_set<string> c;
746         char buf[10];
747
748         clock_t timeStart = clock();
749         for(long i=0; i< value; ++i)
750         {
751             try {
752                 sprintf(buf, 10, "%d", rand());
753                 c.insert(string(buf));
754             }
755             catch(exception& p) {
756                 cout << "i=" << i << " " << p.what();
757                 abort();
758             }
759         }
760         cout << "milli-seconds : " << (clock()-timeStart) << endl; //
761         cout << "unordered_set.size()= " << c.size() << endl;
762         cout << "unordered_set.max_size()= " << c.max_size() << endl;
763         cout << "unordered_set.bucket_count()= " << c.bucket_count() << endl;
764         cout << "unordered_set.load_factor()= " << c.load_factor() << endl;
765         cout << "unordered_set.max_load_factor()= " << c.max_load_factor() << endl;
766         cout << "unordered_set.max_bucket_count()= " << c.max_bucket_count() << endl;
767         for (unsigned i=0; i< 20; ++i) {
768             cout << "bucket #" << i << " has " << c.bucket_size(i) << " elements.\n";
769         }
    }
```

Unordered Set/Multiset:



```
D:\handout\C++11-test\DevC++\VTest-STL\test-stl.exe
select: 15
how many elements: 1000000

test_unordered_set().....
milli-seconds : 2891
unordered_set.size()= 32768
unordered_set.max_size()= 357913941
unordered_set.bucket_count()= 62233
unordered_set.load_factor()= 0.526537
unordered_set.max_load_factor()= 1
unordered_set.max_bucket_count()= 357913941
bucket #0 has 1 elements.
bucket #1 has 1 elements.
bucket #2 has 1 elements.
bucket #3 has 0 elements.
bucket #4 has 3 elements.
bucket #5 has 0 elements.
bucket #6 has 0 elements.
bucket #7 has 1 elements.
bucket #8 has 0 elements.
bucket #9 has 0 elements.
bucket #10 has 0 elements.
bucket #11 has 1 elements.
bucket #12 has 0 elements.
bucket #13 has 2 elements.
bucket #14 has 1 elements.
bucket #15 has 1 elements.
bucket #16 has 1 elements.
bucket #17 has 1 elements.
bucket #18 has 0 elements.
bucket #19 has 0 elements.
target <0^32767>: 23456
::find(), milli-seconds : 0
found, 23456
c.find(), milli-seconds : 0
found, 23456
```

使用容器 `unordered_set`

```
770 string target = get_a_target_string();
771 {
772     timeStart = clock();
773     auto pItem = ::find(c.begin(), c.end(), target); // 比 c.find(...) 慢很多
774     cout << "::find(), milli-seconds : " << (clock() - timeStart) << endl;
775     if (pItem != c.end())
776         cout << "found, " << *pItem << endl;
777     else
778         cout << "not found! " << endl;
779 }
780
781 {
782     timeStart = clock();
783     auto pItem = c.find(target); // 比 ::find(...) 快很多
784     cout << "c.find(), milli-seconds : " << (clock() - timeStart) << endl;
785     if (pItem != c.end())
786         cout << "found, " << *pItem << endl;
787     else
788         cout << "not found! " << endl;
789 }
790
791 }
```

```
D:\handout\C++11-test-DevC++\Test-STL\test-stl.exe
select: 15
how many elements: 1000000

test_unordered_set<>.....
milli-seconds : 2891
unordered_set.size()= 32768
unordered_set.max_size()= 357913941
unordered_set.bucket_count()= 62233
unordered_set.load_factor()= 0.526537
unordered_set.max_load_factor()= 1
unordered_set.max_bucket_count()= 357913941
bucket #0 has 1 elements.
bucket #1 has 1 elements.
bucket #2 has 1 elements.
bucket #3 has 0 elements.
bucket #4 has 3 elements.
bucket #5 has 0 elements.
bucket #6 has 0 elements.
bucket #7 has 1 elements.
bucket #8 has 0 elements.
bucket #9 has 0 elements.
bucket #10 has 0 elements.
bucket #11 has 1 elements.
bucket #12 has 0 elements.
bucket #13 has 2 elements.
bucket #14 has 1 elements.
bucket #15 has 1 elements.
bucket #16 has 1 elements.
bucket #17 has 1 elements.
bucket #18 has 0 elements.
bucket #19 has 0 elements.
target <0x32767>: 23456
::find(), milli-seconds : 0
found, 23456
c.find(), milli-seconds : 0
found, 23456
```



The End

