「They say so」, is half a lie.

– Thomas Fuller

「據他們所說」，這是一半謊言。

– 湯姆斯・博勒

25

標準樣版程式庫

各節標題

本章導讀

除了物件導向觀念在程式碼的用上做出了相當大的貢獻外，另外**泛型化程式設計**（Generic Programming）觀念，亦在程式碼的用上造成革命。達成泛型化程式設計的機制，主要是第 21 章的樣版觀念。雖然，這個觀念並未如物件導向觀念一般，徹底改變程式的**設計思維與運作架構**，但是在處理資料結構上，將泛型化程式設計的觀念與物件導向技術相結合，而建立出的**標準樣版程式庫**（STL，Standard Template Library），仍帶給程式設計師極大的便利。在 21-1-2 節的**範例 21-2** 裡，所示範建立的樣版化鏈結串列類別，就是建立 STL 所運用的技術基礎。

25-1 認識 STL 專家

25-1-1 什麼是 STL

STL！？

標準樣版程式庫簡稱為 STL（Standard Template Library）。其設計的目的，是將程式設計裡經常用到的基本資料結構與演算法，建立為可供程式設計師套用的程式庫。

資料結構與演算法

幾乎所有的電腦程式都被設計用於處理資料，例如：處理實驗數據、統計學生成績、執行複雜的數學運算...等。資料結構是程式內儲存資料之方法，演算法則是處理資料的邏輯。

陣列、鏈結串列與資料結構

第 7 章所介紹的陣列，就是一個最簡單的資料結構，將利用一塊連續的記憶體空間，儲存一連串的資料。但是陣列是一個很不好用的資料結構，尤其是建立矩陣時，一定要先確定這個矩陣到底要有多大。除此之外，在資料的處理上，矩陣也很不好用，當需要在矩陣中間插入一個資料時，這幾乎是很...令人難過的事。

而 21-1-1 節所介紹的鏈結串列，則是一個比較複雜的資料結構。在資料的儲存和處理上，顯然比陣列好許多，因為它不需要在建立時，就確定大小，也可以在鏈結串列中間插入一個節點。

25-1-2 使用 STL 的效益

說了半天，您對 STL 的觀念相信仍然是模糊的。您可以回想一下 21-1-1 節所介紹鏈結串列的觀念，以及**範例 21-1** 所示範鏈結串列的建立。而 21-1-2 節的**範例 21-2**，則利用樣版類別的觀念，讓不同的資料型態都能使用 ListNode 類別。

而 STL 的一部份功能，就是提供程式設計師如同**範例 21-2** 的 ListNode 樣版類別，讓程式設計師不必自行設計，只需直接套用即可執行資料處理，這大大減輕了程式設計師的負擔。有鑑於此，C++標準便將 STL 納入其中。而 STL 的建立所利用的技術與觀念，就是在第 21 章所提到的**樣版**觀念。

STL 的組成

STL 由以下五個部份所組成：

1. **容器**（container）介紹於 25-2 節、25-3 節、25-4 節
2. **指位器**（iterator）將與容器一併說明
3. **演算法**（algorithm）說明於 25-5 節
4. **函數物件**（function object）說明於 25-6 節
5. **轉接器**（adaptor）說明於 25-7 節

25-2 容器與指位器 專家

25-2-1 容器

什麼是...

STL 裡提供了許多種容器，每一種容器就是一種儲存資料的方法，比如 List 容器，就是 STL 所提供的鏈結串列。而 21-1-2 節範例 21-2 的 LinkNode 樣版類別就像是一個簡化的 STL 之 List 容器。所以，您就不難瞭解，STL 的容器就是一個已經建立完成的資料結構。您可以在這個容器裡，放入任何型態的資料，甚至是自行定義的類別，便可輕易建立出一個儲存該型態（類別）的資料結構。

容器的種類

在 STL 的容器大致可分為序列容器（Sequence Container）、關聯容器（Associate Container）兩種。每種容器都有其個別的方法，用於操作容器，比如：新增、刪除或取得元素之個數。以下將大略介紹上述兩種容器，在以後各節裡，則將為您進一步說明使用語法以及各容器方法。

一、序列容器（Sequence Container）

我們可以把序列容器儲存資料的方式，想像成一條直線，而資料就儲存在這條假想的直線上，陣列與鏈結串列儲存資料的方式就是這樣。由於是以線狀的方式儲存資料，當將資料儲存至序列時，每個元素的先後順序將被確定。在 STL 裡，序列容器共有以下三種特性：

容器名稱	特性
vector	可以當做一種智慧型的陣列，在記憶體中佔有一塊連續的空間，可變動大小。適用於由後端增加資料。可用『[ ]』運算子直接存取資料。在前端與中間插入資料的速度較為緩慢。
deque	與 vector 類似，適用於由兩端插入資料。在記憶體中，並不佔有連續的空間。可用『[ ]』運算子直接存取資料。在中間插入資料的速度較為緩慢。
list	雙向鏈結串列，每個元素均具有指標指向前一個與下一個資料。從串列任何一處插入資料都很迅速，不支援以『[ ]』運算子直接存取資料。

二、關聯容器（Associate Container）

關聯容器與序列容器不同的地方，在於關聯容器並不是以線狀的方式儲存資料。當將元素插入容器時，各元素間並不以插入的先後或位置做為順序，而以**關鍵值**（key）做為元素順序的依據。所以，元素被插入容器裡時，元素的關鍵值將會依照某一順序排列。若是數字則依照數字的大小排列，若是字串則依照英文字母的順序排列。在 STL 裡，將提供以下四種關聯容器。

容器名稱	特性
set	只儲存 key，不儲存對應值，且 key 不可重複。
multiset	只儲存 key，不儲存對應值，但 key 可重複。
map	儲存 key 與對應值，且 key 不可重複。
multimap	儲存 key 與對應值，但 key 可重複。

25-2-2 指位器

什麼是...

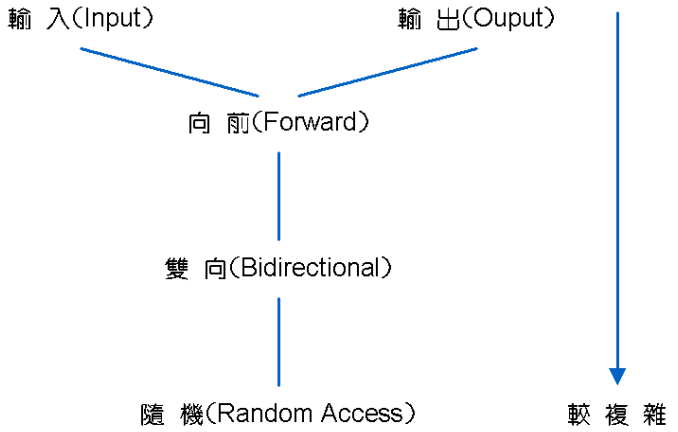
指位器（Iterator）是容器使用的**特殊指標**，透用指位器可以存取容器內儲存的資料。而 STL 中每一種容器，都分別定義了可操作容器的指位器。

指位器的種類

指位器的種類有輸入（Input）、輸出（Output）、向前（Forward）、雙向（Bidirectional）以及隨機存取（Random Access）五種。請注意！輸入與輸出是站在使用容器之程式的角度，所以，輸入是指在程式內從容器讀取元素資料，輸出則是從程式將資料寫入容器的元素。各指位器的功能說明如下：

指位器名稱	功能
輸入（Input）	用於讀取容器內的資料。其移動方向，是由容器的第一個資料向最後一筆資料移動，也就是向前移動，且一次移動一個元素。
輸出（Output）	用於將資料寫入容器。其移動方向，是由容器的第一個資料向最後一筆資料移動，也就是向前移動，且一次移動一個元素。
向前（Forward）	包含輸出/入指位器的功能，可用於容器元素的輸出/入。
雙向（Bidirectional）	包含向前指位器的功能，且可向後移動。
隨機存取（Random Access）	包含雙向指位器的功能，且可直接跳到容器的某一個元素。

下圖為這五種指位器的架構圖，由上而下指位器的複雜度遞增，且圖下方的指位器支援圖上方指位器的功能。請注意！指位器間並沒有繼承關係。



25-2-3 容器與指位器

每一種容器都定義了自己的指位器，下表將說明各容器指位器所支援的功能。

類型	輸出	輸入	向前	雙向	隨機存取
vector	*	*	*	*	*
list	*	*	*	*	
deque	*	*	*	*	*
set	*	*	*	*	
multiset	*	*	*	*	
map	*	*	*	*	
multimap	*	*	*	*	

指位器的使用

下表中列出，各種指位器所能執行的運算。

運算（p 代表指位器）	說明
向前指位器	
p++、++p	逐元素向前移動指位器
輸入指位器	
*p	讀取指位器所指向的元素值，用於等式的右邊。
p = p1	將 p1 指向的位址設定給 p
輸出指位器	
*p	寫入指位器所指向的元素值，用於等式的左邊。
p = p1	將 p1 指向的位址設定給 p
p == p1	比較 p 與 p1 是否相等
p != p1	比較 p 與 p1 是否不相等

運算（p 代表指位器）	說 明
雙向指位器	
p--、--p	逐元素向後移動指位器
隨機存取指位器	
p[i]	容器內第 i 個元素
p + i	將指位器向前移動 i 個元素
p - i	將指位器向後移動 i 個元素
p > p1	若 p 指向元素在 p1 之前，則傳出 true，否則傳出 false。
p >= p1	若 p 指向元素等於 p1 或者在 p1 之前，則傳出 true，否則傳出 false。
p < p1	若 p 指向元素在 p1 之後，則傳出 true，否則傳出 false。
p <= p1	若 p 指向元素等於 p1 或者在 p1 之後，則傳出 true，否則傳出 false。
p += I	指位器向前移動 i 個元素
p -= I	指位器向後移動 i 個元素

25-3 序列容器 專家

25-3-1 vector

vector 和矩陣是差不多的東西，兩者儲存在記憶體中都佔有一塊連續的空間。只是在宣告 vector 時，並不需要像宣告陣列那樣，必須先確定陣列的大小。當有資料要存入 vector 時，可以直接從尾端增加，vector 將自動配置記憶體空間。但當該塊連續記憶體空間無法再增加元素時，整個容器將被搬移到可以容納該容器的連續記憶體空間。所以，vector 可以說是一種可變動大小的智慧型陣列。

vector 容器在記憶體中儲存方法的示意

使用 vector

使用 vector 時，必須載入 vector 標頭檔。

```
#include <vector>
using namespace std;
```

宣告 vector 容器的語法如下：

vector<資料型態> 變數名稱;

語法中各部份的用途說明如下：

- ✧ 資料型態
  vector 容器內欲儲存資料的型態。
- ✧ 變數名稱
  運用 vector 容器宣告之物件名稱。

以下敘述將宣告 int_v 為容納 int 型態資料的 vector 容器。

```
vector<int> int_v;  //宣告一個容納整數型態資料的 vector 容器
```

vector 的方法

下表將列出 vector 容器內較為重要的成員函數。

名稱	用途
插入與刪除	
push_back	在容器後端增加元素
pop_back	在容器後端刪除元素
insert	在容器中間插入元素
erase	刪除容器中間的元素
clear	清除容器內的元素

名稱	用途
取出器	
front	傳回容器前端元素的參考
back	傳回容器末端元素的參考
begin	傳回容器前端的指位器
end	傳回容器末端的指位器
rbegin	傳回容器前端的倒轉指位器，相關運用請參考 25-7-5 節。
rend	傳回容器末端的倒轉指位器，相關運用請參考 25-7-5 節。
max_size	傳回容器可儲存元素的最大個數
size	傳回目前容器中的元素個數
empty	若容器內目前無元素傳回 true，反之傳回 false
capacity	傳回目前容器不需重新配置記憶體，所可儲存之最大元素個數。
at(n)	傳回第 n 個元素的參考
運算子	
operator[]	利用[]運算子取出容器中的元素
其他	
swap(x)	與 x（vector 容器）互換容器內的元素

指位器的宣告與使用

在使用 vector 容器的指位器前，必須先完成宣告。以下敘述將宣告 i 為指向 vector<int>型態容器的指位器。

```
typename vector<int>::iterator i;
//宣告一個 vector<int>容器使用的指位器
```

以上敘述宣告指位器前，將以 typename 定義『vector<int>::iterator』為一種型別，以避免編譯器誤認為 vector<int>類別的 iterator 靜態資料成員。

以下敘述將示範指位器配合前面提到的**容器取出器**，取出容器內元素的方法。

```
vector<int> con;
typename vector<int>::iterator i;  //宣告指位器

//利用 for 迴圈列印 container 元素
for(i = con.begin(); i != con.end(); i++)
{  cout << *i << " ";} //利用指位器取得 container 內的元素
```

以上敘述將 i 宣告為指向『vector<int>』容器的指位器，並利用 con 容器的 begin 與 end 取出器，傳出 con 容器內頂端與尾端的元素，做為 for 迴圈的上界與下界，並利用 cout 輸出 i 指位器指向元素的值（*i）。以下範例將示範 vector 容器各成員函數的使用。

範例 25-1：vector 的使用

[執行結果]

```
Push back 1, 6, 4 ...
1 6 4
Insert 9 ...
v :9 1 6 4
Erase second element ...
v : 9 6 4
Pop back ...
v : 9 6
Second element : 6
The Size of Vector : 2
Maximum size : 1073741823
Using array to initialize vector ...
v1 : 1 4 3 2
Assgin v1 to v ...
v : 1 4 3 2
請按任意鍵繼續 . . .
```

[程式碼]

檔案位置：ex25-1\stl_vector.cpp

```
001  /*
002  範例檔名：stl_vector.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
```

```
006   #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007   #include <vector> //載入 vector 標頭檔
008
009   using namespace std; //使用 std 名稱空間
010
011   #define Size 4
012
013   template <class T>  //宣告用於列印的樣版函數原型
014   void print(vector<T> &);
015
016   int main() //主程式開始 <== 程式進入點
017   {
018      vector<int> v;//宣告儲存整數用的 vector
019
020      cout << "Push back 1, 6, 4 ..." << endl;
021      v.push_back(1);  //從 vector 後端增加元素
022      v.push_back(6);
023      v.push_back(4);
024      print(v);   //輸出容器內的資料
025
026      cout << "Insert 9 ..." << endl << "v :";
027      v.insert(v.begin(),9); //在 vector 前端插入 9
028      print(v);
029
030      cout << "Erase second element ..." << endl << "v : ";
031      v.erase(v.begin()+1);  //刪除 vector 前端的資料
032      print(v);
033
034      cout << "Pop back ..." << endl << "v : ";
035      v.pop_back(); //刪除 vector 末端的資料
036      print(v);
037
038      cout << "Second element : " << v[1] << endl;  //利用[]存取元素
039      cout << "The Size of Vector : " << v.size() << endl;
040      //輸出 vector 的元素個數
041      cout << "Maximum size : " << v.max_size() << endl;
042      //輸出可容納的元素個數
043
044      cout << "Using array to initialize vector ..." << endl << "v1 : ";
045      int a[Size] = {1,4,3,2};
046      vector<int> v1(a, a+Size); //利用陣列做 vector 的初值設定
047      print(v1);
048
```

```
049      cout << "Assgin v1 to v ..." << endl << "v : ";
050      v = v1;  //將 v 指派給 v1
051      print(v);
052
053      system("PAUSE");
054      return 0;
055   } //主程式結束
056
057   template <class T>
058   void print(vector<T> & pri_con)     //用於列印容器內容的樣版函數
059   {
060      if(pri_con.empty()){   //判別 pri_con 是否有資料
061         cout << "Container is empty!" << endl;}
062      else{
063         typename vector<T>::iterator i;   //宣告指位器
064
065          //利用 for 迴圈列印 container 元素
066         for(i = pri_con.begin(); i != pri_con.end(); i++)
067         {
068            cout << *i << " ";     //利用指位器取得 container 內的元素
069         }
070         cout << endl;
071      }
072   }
```

[程式說明]

第 7 行：使用 vector 容器前，必須先載入 vector 標頭檔。

第 18 行：利用『vector<int>』將 v 宣告為容納 int 資料的 vector 容器。

第 21～23 行：利用 push_back()將整數 1、6、4 放進容器裡。

第 27 行：利用 insert()將 9 插入到容器的頂端，其中 insert 的第一個參數將傳入欲插入元素之位置的指位器，第二個參數則是欲插入的資料，我們將利用『v.begin()』傳回 v 容器頂端元素的指位器，並插入 9。

第 31 行：利用『v.begin()+1』傳回 v 容器第 2 個元素的指位器，並利用 erase 刪除元素資料。

第 35 行：呼叫 pop_back()刪除 vector 容器末端的元素。

第 38 行：利用『[ ]』運算子，透過索引取得容器內的元素，索引的初值為 0。

第 39 行：利用 size()取得 vector 容器內的元素個數。

第 41 行：利用 max_size()取得 vector 容器內，可容納元素的最多個數。

第 45、46 行：宣告一個 a 陣列，並利用 a 陣列做為初值設定 v1 容器，傳入的參數分別是陣列首尾的位址值。

第 50 行：利用『=』運算子將 v1 指派給 v。

第 57～72 行：宣告 print 樣版函數，用於輸出容器內的元素。但必須注意的是，由於樣版函數內使用了 empty()、begin()與 end()成員函數，因此，傳入的容器必須支援這三個成員函數。

第 66～69 行：利用 for 迴圈與 begin()、end()方法，輸出容器內的元素。

25-3-2 deque

deque 的特性基本上與 vector 類似，只是 deque 可以從前端以 push_front()方法新增元素，且儲存時，不需要運用一塊連續的記憶體空間儲存整個容器的元素。當宣告一個 deque 元素後，若不斷增加容器裡的元素，將導致原先儲存宣告容器之記憶體空間不足時，deque 容器將不會把整個容器搬移到另一個連續的記憶體空間，而是將新增的元素儲存到另一塊記憶體去。所以，整個容器在記憶體裡，可能被分成若干段儲存。

deque 容器在記憶體中儲存方式的示意圖

使用 deque 容器

使用 deque 時，必須載入 deque 標頭檔。

```
#include <deque>
using namespace std;
```

宣告一個 deque 容器的語法如下：

deque <資料型態> 變數名稱;

語法中各部份的用途說明如下：

- ✧ 資料型態
  deque 容器內欲儲存資料的型態。
- ✧ 變數名稱
  宣告為 deque 容器之物件名稱。

以下敘述將宣告一個容納 int 資料的 deque 容器。

```
deque<int> int_d;    //宣告容納 int 的 deque 容器
```

deque 容器的方法

下表將列出 deque 容器中較為重要的成員函數。

名稱	用途
插入與刪除	
push_front	在容器前端增加元素
pop_front	在容器前端刪除元素
push_back	在容器後端增加元素
pop_back	在容器後端刪除元素
insert	在容器中間插入元素
erase	刪除容器中間的元素
clear	清除容器內的元素

名稱	用途
取出器	
front	傳回容器前端元素的參考
back	傳回容器末端元素的參考
begin	傳回容器前端的指位器
end	傳回容器末端的指位器
rbegin	傳回容器前端的倒轉指位器，請參考 25-7-5 節的說明。
rend	傳回容器末端的倒轉指位器，請參考 25-7-5 節的說明。
max_size	傳回容器可儲存元素的最大個數
size	傳回目前容器中的元素個數
empty	若容器內目前無元素傳回 true，反之傳回 false
at(n)	傳回第 n 個元素的參考
運算子	
operator[]	利用 []運算子取出容器中的元素
其他	
swap(x)	與 x（vector 容器）互換容器內的元素

指位器的宣告與使用

以下敘述將宣告指向『deque<int>』型態容器的指位器。

```
typename deque <int>::iterator i;
//宣告一個 deque <int>容器使用的指位器
```

以上敘述宣告指位器前，必須以 typename 定義『deque<int>::iterator』是一個型別名稱。以下範例將示範 deque 容器各成員函數的使用。

範例 25-2：deque 的使用

[執行結果]

```
Push back 1, 6 and push front 4 ...
q : 4 1 6
Insert 9 into 2nd element ...
```

```
q : 4 9 1 6
Erase second element ...
q : 4 1 6
Pop back ...
q : 4 1
The Size of deque : 2
Maximum size : 4294967295
Using another deque to initialize deque ...
q1 : 4 1
請按任意鍵繼續 . . .
```

[程式內容]

檔案位置：ex25-2\stl_deque.cpp

```
001  /*
002  範例檔名：stl_deque.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <deque> //載入 deque 標頭檔
008
009  using namespace std; //使用 std 名稱空間
010
011  template <class T>
012  void print(deque<T> & pri_con)//用於列印容器內容的樣版函數
013  {
014     if(pri_con.empty()) //判別 pri_con 是否有資料
015        cout << "Container is empty!" << endl;
016     else{
017        typename deque<T>::iterator i;    //宣告指位器
018
019         //利用 for 迴圈列印 container 元素
020        for(i = pri_con.begin(); i != pri_con.end(); i++)
021        {
022           cout << *i << " ";    //利用指位器取得 container 內的元素
023        }
024        cout << endl;
025     }
026  }
027
028  int main() //主程式開始 <-- 程式進入點
029  {
```

```
030      deque<int> q; //宣告儲存整數用的 deque
031
032      cout << "Push back 1, 6 and push front 4 ..." << endl << "q : ";
033      q.push_back(1);  //從 deque 後端增加元素
034      q.push_back(6);
035      q.push_front(4);
036      print(q);
037
038      cout << "Insert 9 into 2nd element ..." << endl << "q : ";
039      q.insert(q.begin()+1,9);    //將 9 插入第 2 元素之後
040      print(q);
041
042      cout << "Erase second element ..." << endl << "q : ";
043      q.erase(q.begin()+1);  //刪除第 2 個元素之後的資料
044      print(q);
045
046      cout << "Pop back ..." << endl << "q : ";
047      q.pop_back(); //刪除 deque 末端的資料
048      print(q);
049
050      cout << "The Size of deque : " << q.size() << endl;
051      //輸出 deque 的元素個數
052      cout << "Maximum size : " << q.max_size() << endl;
053      //輸出可容納的元素個數
054
055      cout << "Using another deque to initialize deque ..."
056           << endl << "q1 : ";
057      deque<int> q1 = q;  //利用另一個 deque 容器做初值設定
058      print(q1);
059
060      system("PAUSE");
061      return 0;
062   } //主程式結束
```

[程式說明]

第 11～26 行：定義 print()樣版函數，與**範例 25-1** 的 print()樣版函數的差異僅在操作容器的類型（第 12、17 行）。

第 33～35 行：利用 push_front()與 push_back()新增元素至容器中。

第 39 行：利用 insert()將 9 插入到容器的第 2 個元素後。其中 insert()的第一個參數將傳入欲插入元素之位置的指位器，第二個參數則是欲插入的資料，將利用『q.begin() + 1』傳回 q 容器第 2 個元素的指位器，並插入 9。

第 43 行：利用『q.begin() + 1』傳回 v 容器第 2 個元素的指位器，並利用 erase()刪除元素資料。

第 47 行：利用 pop_back()移除容器尾端的元素。

第 50 行：利用 size()取得 vector 容器內的元素個數。

第 52 行：利用 max_size()取得 vector 容器內，可容納元素的最多個數。

第 57 行：示範宣告 deque 容器時，以另一 deque 容器做為初值的宣告方法。其他與 vector 相同之成員函數的使用，請參考 25-3-1 節**範例 25-1**。

25-3-3 list

list 容器是 STL 提供的**雙向**鏈結串列，每個元素都知道其前一個與下一個元素，而在 21-1-1 節介紹的則是**單向**鏈結串列。

使用 list 容器

使用 list 時，必須載入 list 標頭檔。

```
#include <list>
using namespace std;
```

宣告 list 容器的語法如下：

list <資料型態> 變數名稱;

以下敘述將宣告容納 int 資料的 list 容器。

```
list <int> int_l;   //宣告容納 int 的 list 容器
```

list 容器的方法

下表將列出 list 容器內較為重要的成員函數。

名稱	用途
插入與刪除	
push_front	在容器前端增加元素
pop_front	在容器前端刪除元素
push_back	在容器後端增加元素
pop_back	在容器後端刪除元素
insert	在容器中間插入元素
erase	刪除容器中間的元素
clear	清除容器內的元素
取出器	
front	傳回容器前端元素的參考
back	傳回容器末端元素的參考
begin	傳回容器前端的指位器
end	傳回容器末端的指位器
rbegin	傳回容器前端的倒轉指位器，請參考 25-7-5 節的說明。
rend	傳回容器末端的倒轉指位器，請參考 25-7-5 節的說明。
max_size	傳回容器可儲存元素的最大個數
size	傳回目前容器中的元素個數
empty	若容器內目前無元素傳回 true，反之傳回 false。
其他	
swap(x)	將 x（list 容器）互換容器內的元素
sort	將元素重新以遞增方式排序
merage(x)	將 x（list 容器）的內容與容器結合

指位器的宣告與使用

以下敘述將宣告一個指向『list <int>』型態容器的指位器。

```
typename list <int>::iterator i; //宣告一個 list<int>容器使用的指位器
```

以上敘述宣告指位器前，必須以 typename 定義『list<int>::iterator』是一個型別名稱。以下範例將示範 list 容器各成員函數的使用。

範例 25-3：list 的使用

[執行結果]

```
Push back o, k, e and push front J ...
J o k e
Pop front ...
L : o k e
Pop back ...
L : o k
First Element of L : o
Last Element of L : k
Maximum size of L : 4294967295
Insert array to list ...
L2 : M a r y
Merge L and L2 ...
L : M a o k r y
L2 : Container is empty!
Assign L to L2 ...
L2 : M a o k r y
Reverse L2 ...
L2 : y r k o a M
Sort L2 ...
L2 : M a k o r y
請按任意鍵繼續 . . .
```

將 Mary 插入容器內

[程式內容]

檔案位置：ex25-3\stl_list.cpp

```
001   /*
002   範例檔名：stl_list.cpp
003   程式開發：郭尚君
004   */
```

```
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <list>  //載入 list 標頭檔
008
009  using namespace std; //使用 std 名稱空間
010
011  #define Size 4
012
013  template <class T>
014  void print(list<T> & pri_con) //用於列印容器內容的樣版函數
015  {
016     if(pri_con.empty()) //判別 pri_con 是否有資料
017        cout << "Container is empty!" << endl;
018     else{
019        typename list<T>::iterator i; //宣告指位器
020
021         //利用 for 迴圈列印 container 元素
022        for(i = pri_con.begin(); i != pri_con.end(); i++)
023        {
024           cout << *i << " ";    //利用指位器取得 container 內的元素
025        }
026        cout << endl;
027     }
028  }
029
030  int main() //主程式開始  程式進入點
031  {
032     list<char> L; //宣告儲存整數用的 list
033
034     cout << "Push back o, k, e and push front J ..." << endl;
035     L.push_back('o');   //從 list 後端增加元素
036     L.push_back('k');
037     L.push_back('e');
038     L.push_front('J');  //從 list 前端增加元素
039     print(L);
040
041     cout << "Pop front ..." << endl << "L : ";
042     L.pop_front();//刪除 list 前端的資料
043     print(L);
044
045     cout << "Pop back ..." << endl << "L : ";
046     L.pop_back(); //刪除 list 末端的資料
047     print(L);
```

```
048
049       cout << "First Element of L : " << L.front() << endl;
050        //顯示前端的元素
051       cout << "Last Element of L : " << L.back() << endl;
052       //顯示尾端的元素
053       cout << "Maximum size of L : " << L.max_size() << endl;
054       //輸出可容納的元素個數
055
056       cout << "Insert array to list ..." << endl << "L2 : ";
057       char a[Size] = {'M','a','r','y'};
058       list<char> L2;
059       L2.insert(L2.begin(),a, a+Size); //將陣列插入 list
060       print(L2);
061
062       cout << "Merge L and L2 ..." << endl << "L : ";
063       L.merge(L2); //將 L2 結合至 L
064       print(L);
065       cout << "L2 : ";
066       print(L2);
067
068       cout << "Assign L to L2 ..." << endl <<"L2 : ";
069       L2 = L;  //將 L 指派給 L2
070       print(L2);
071
072       cout << "Reverse L2 ..." << endl << "L2 : ";
073       L2.reverse(); //將 L2 的元素倒轉
074       print(L2);
075
076       cout << "Sort L2 ..." << endl << "L2 : ";
077       L2.sort();  //將 L2 的元素重新排序
078       print(L2);
079
080       system("PAUSE");
081       return 0;
082    } //主程式結束
```

[程式說明]

第 59 行：利用 insert()成員函數將陣列插入 list 中。insert()的第一個參數將傳入一個指位器，表示陣列將插入該指位器指向元素的位置。第二個參數為欲插入陣列的起始指標。第三個是陣列尾端的指標。範例內宣告一個 a 陣列，並將該陣列插入 L2 的第一個元素之後（begin()將傳回指向第一個元素的指位器）。

25-4 關聯容器 專家

25-4-1 set 與 multiset

set 與 multiset 中儲存的資料，被稱為該容器的索引值（key）。而 set 與 multiset 的不同，在於 set 不允許索引值重複，但 multiset 容許。而在使用語法與兩者的成員上，兩者大致上也是相同，因此，本節的程式範例將代表性地示範 set 容器的使用。

使用 set 與 multiset 容器

使用 set 與 multiset 時，必須載入 set 標頭檔。

```
#include <set>
using namespace std;
```

宣告一個 set 與 multiset 容器的語法如下：

```
set <儲存資料型態, 函數物件<資料型態> > 變數名稱;
```

這裡要插入一個空白，否則會被編譯器誤認為>>運算子

或

```
multiset <儲存資料型態, 函數物件<資料型態> > 變數名稱;
```

語法中各部份的用途說明如下：

- ✧ 儲存資料型態

  set 或 multiset 容器內欲儲存資料的型態。

- ✧ 函數物件

  set 或 multiset 容器內執行排序的函數物件名稱，『函數物件<資料型態>』敘述若省略，將使用預設的 less 函數物件。函數物件介紹於 25-6 節。

- ✧ 資料型態

  函數物件執行資料排序時，排序資料的型態。

- ✧ 變數名稱

  運用 set 或 multiset 容器宣告之物件名稱。

請注意！在語法內，最後一個『>』與倒數第二個『>』中間必須插入一個空白，否則編譯器會把它們誤認為『>>』運算子。以下將宣告 s 為容納 string 型態的 set 容器。


```
set<string, less<string> > s;     //將 s 定義爲容納字串的 set 容器
```

set 與 multiset 容器的方法

下表將列出 set 與 multiset 容器內較為重要的成員函數。

名稱	用途
插入與刪除	
insert	在容器中間插入元素
erase	刪除容器中間的元素
clear	清除容器內的元素
取出器	
begin	傳回容器前端的指位器
end	傳回容器末端的指位器
rbegin	傳回容器前端的倒轉指位器，請參考 25-7-5 節的說明。
rend	傳回容器末端的倒轉指位器，請參考 25-7-5 節的說明。
max_size	傳回容器可儲存元素的最大個數
size	傳回目前容器中的元素個數
empty	若容器內目前無元素傳回 true，反之傳回 false
find	尋找某一元素
lower_bound	尋找某一元素，並以指標為下界
upper_bound	尋找某一元素，並以指標為上界
其他	
swap(x)	將 x 與自己互換

typedef 的使用

當使用容器宣告變數時，容器的型態往往相當的長，此時，可利用 typedef（10-1 節），將容器的型態定義成另一個自訂的型態名稱。以下敘述將把『set<string, less<string> >』定義為 str_set。

```
typedef set<string, less<string> > str_set;
```

範例 25-4：set 的使用

[執行結果]

```
Ann Joe John Ken Mary
Ann Joe John Judy Ken Mary
Please input name for searching :
Lee <-------- 欲尋找的字串
The Lee is not in container !
Please input a range for searching :(Upper/Lower)
Lee <-------- 尋找範圍的上界
Bill <-------- 尋找範圍的下界
Joe  John  Judy  Ken
請按任意鍵繼續 . . .
```

[程式內容]

檔案位置：ex25-4\stl_set.cpp

```
001  /*
002  範例檔名：stl_set.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <string>
008  #include <set>//載入 set 標頭檔
009
010  using namespace std; //使用 std 名稱空間
011
012  typedef set<string, less<string> > str_set;   <-- 插入一個空白
013  //定義 str_set 代表 set<string, less<string> >
014
015  template <class T>
016  void print(set<T> & pri_con)  //用於列印容器內容的樣版函數
```

```
017   {
018      if(pri_con.empty()) //判別 pri_con 是否有資料
019         cout << "Container is empty!" << endl;
020      else{
021         typename set<T>::iterator i; //宣告指位器
022
023          //利用 for 迴圈列印 container 元素
024         for(i = pri_con.begin(); i != pri_con.end(); i++)
025         {
026            cout << *i << " ";    //利用指位器取得 container 內的元素
027         }
028         cout << endl;
029      }
030   }
031
032   int main() //主程式開始  程式進入點
033   {
034      string name[] = {"Mary","John","Ann","Joe","Ken"};
035      str_set s(name, name + 5); //利用字串陣列設定 set 容器內的元素
036      print(s);  //列印容器內的元素
037
038      s.insert("Judy");   //插入元素
039      print(s);
040
041      string search, upper, lower;
042      str_set::iterator iter;     //宣告指位器
043
044      cout << "Please input name for searching : " << endl;
045      cin >> search;
046
047      iter = s.find(search); //尋找 search 字串，若無則傳回尾端元素的指位器
048      if(iter == s.end())
049         cout << "The " << search << " is not in container !" << endl;
050      else
051         cout << "The " << search << " is in container !" << endl;
052
053      cout << "Please input a range for searching :(Upper/Lower)"
054           << endl;
055      cin >> upper >> lower; //讀取範圍的上界與下界
056
057       //輸出在尋找範圍內的字串
058      for(iter = s.lower_bound(lower)
059                                ; iter != s.upper_bound(upper); iter++)
```

```
060       {
061          cout << *iter << "  ";    //輸出指位器指向的元素值
062       }
063       cout << endl;
064
065       system("PAUSE");
066       return 0;
067    } //主程式結束
```

[程式說明]

第 12 行：利用 typedef 將『set<string, less<string> >』定義為 str_set 的自訂型態。

第 34 行：定義 name 字串陣列，並設定初值。

第 35 行：運用字串陣列建立 set 物件。

第 42 行：宣告指位器。

第 47 行：利用 find()成員函數尋找容器裡，內容符合欲搜尋字串之元素。若找不到，則將傳回指向容器尾端元素之指位器。

第 48 行：比對 find()成員函數傳回的指位器，是否與 end 成員函數傳回的指位器相同，若是，則代表 iter 指位器將指向容器內最後一個元素。

第 53～62 行：利用輸入的 upper 字串與 lower 字串，配合 upper_bound()與 lower_bound()成員函數，尋找排列順序介於 upper 與 lower 間的字串。從執行結果可以看出，輸入 Lee 為上界，Bill 為下界，將可尋找出介於兩字串順序間的字串。

25-4-2 map 與 multimap

在 map 與 multimap 中，除了儲存索引值（key）外，還儲存對應的資料，如：物件或者字串。這點與 set（multiset）不同，而 map 與 multimap 的差異，也僅在於索引值是可否重複。

使用 map 與 multimap 容器

使用 map 與 multimap 時，必須載入 map 標頭檔。

```
#include <map>
using namespace std;
```

宣告一個 map 與 multimap 容器的語法如下：

```
map <索引值型態, 儲存資料型態, 函數物件<資料型態> > 變數名稱;
```

這裡要插入一個空白，否則會被編譯器誤認為>>運算子

或

```
multimap <索引值型態, 儲存資料型態, 函數物件<資料型態> > 變數名稱;
```

語法各部分說明如下：

- ✧ 索引值型態
  map 或 multimap 容器內用於查詢資料之索引值的型態。
- ✧ 儲存資料型態
  map 或 multimap 容器內儲存資料的型態。
- ✧ 函數物件
  map 或 multimap 容器內執行排序的函數物件名稱，語法內『函數物件<資料型態>』可省略，使用預設的 less 函數物件。函數物件說明於 25-6 節。
- ✧ 資料型態
  函數物件執行資料排序時，排序資料的型態。
- ✧ 變數名稱
  宣告為 set 或 multiset 容器之物件的名稱。

請注意！在上述語法內，最後一個『>』與倒數第二個『>』中間必須插入一個空白，否則編譯器會把它們誤認為『>>』運算子。以下敘述將宣告 m 物件為容納 string 型態的 map 容器。

```
map<string, string, less<string> > m; //將 m 定義爲容納字串的 map 容器
```

map 與 multimap 容器的方法

下表將列出 map 與 multimap 容器中較為重要的成員函數。

名稱	用途
插入與刪除	
insert	在容器中間插入元素
erase	刪除容器中間的元素
clear	清除容器內的元素
取出器	
begin	傳回容器前端的指位器
end	傳回容器末端的指位器
rbegin	傳回容器前端的倒轉指位器，請參考 25-7-5 節的說明。
rend	傳回容器末端的倒轉指位器，請參考 25-7-5 節的說明。
max_size	傳回容器可儲存元素的最大個數
size	傳回目前容器中的元素個數
empty	若容器內目前無元素傳回 true，反之傳回 false。
find	尋找某一元素
lower_bound	尋找某一元素，並以其為下界。
upper_bound	尋找某一元素，並以其為上界。
其他	
swap(x)	將 x 與自己互換

範例 25-5：map 的使用

[執行結果]

```
Bill, A junior high school student.
John, A senior high school student.
Mary, A senior high school teacher.
請按任意鍵繼續 . . .
```

[程式碼]

檔案位置：ex25-5\stl_map.cpp

```
001  /*
002  範例檔名：stl_map.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <string>
008  #include <map>//載入 map 標頭檔
009
010  using namespace std; //使用 std 名稱空間
011
012  int main() //主程式開始 <== 程式進入點
013  {
014     typedef map<string, string, less<string> > map_type;
015     //利用 typedef 定義新的型態
016
017     map_type m;//宣告容器
018     map_type::iterator iter; //宣告指位器
019
020     //插入資料
021     m.insert(map_type::value_type("John"
                        ,"A senior high school student."));
022     m.insert(map_type::value_type("Mary"
                        ,"A senior high school 023   teacher."));
023     m.insert(map_type::value_type("Bill"
                        ,"A junior high school student."));
024
025     for(iter = m.begin(); iter != m.end(); iter++)
026        cout << (*iter).first << ", " << (*iter).second << endl;
027        //輸出關鍵值與對應之資料
028
029      system("PAUSE");
030      return 0;
031  } //主程式結束
```

[程式說明]

第 21～23 行：呼叫 insert()成員函數將資料新增到 map 容器內。由於 map 與 multimap 容器的元素，是由關鍵值與資料兩個部份組成。但 insert()

成員函數只有一個參數，因此，必須利用『map_type::value_type』將兩個資料，組合成一組資料新增到容器中。這種由兩個資料組成一組資料的資料組稱為 pair。在 map 的標頭檔裡，將把 value_type 定義成一個 pair。

```
map_type::value_type("John"    ,"A senior high school student.")
```

所以，傳入 insert()成員函數的參數，實際上是由 map 所定義的 pair 物件。取出 pair 物件時，將利用 first()與 second()傳出 pair 內的資料。

第 25、26 行：利用 begin()與 end()成員函數，取得 map 容器指向第一個元素與最後一個元素的指位器，再透過 first()與 second()從 pair 物件，取出索引值與對應的資料。

25-5 演算法 專家

25-5-1 演算法的介紹

什麼是演算法

演算法（Algorithm）是 STL 已經完成撰寫，提供給程式設計師使用的資料處理函數，這些函數建立的方法，將利用樣版函數的技術完成。STL 中提供的演算法相當多，如：資料的尋找、排序、比對、複製...等。透過這些樣版函數，程式設計師可以很容易地運用演算法，操作容器內儲存的資料。

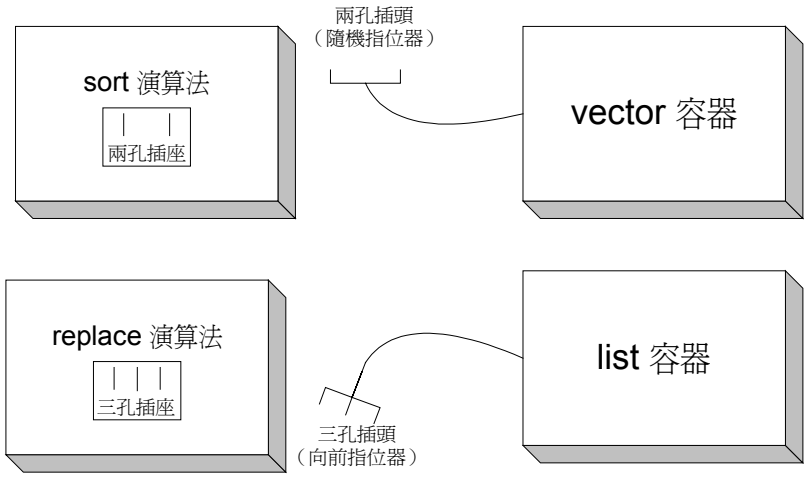
容器、指位器與演算法

在每一個容器的定義裡，均將定義適合該容器使用的指位器，且不同容器的指位器能夠執行的計算也不同。在 25-2-3 節**容器與指位器**的介紹裡，均將利用一個表格，詳細說明每一種容器指位器所支援的指位器型態，並在同節的**指位器的使用**中，告訴各位每一種指位器可以執行的計算有哪些。

各種指位器所能支援的指位器型態，與該容器的儲存方式有很大的關係，比如：vector 是儲存在一塊連續的記憶體中，對於容器內的元素，可以透過指標運算的方式直接存取，所以 vector 的指位器可以隨機存取。但是 list 的指位器就不是儲存在連續的記憶體中，因此，要讀取 list 的第三個元素，必須從第一個元素開始，一個一個往下讀，無法直接存取容器的任一個元素，所以 list 的指位器無法隨機存取。

然而 STL 的演算法在設計時，被設計成與容器無關，所以需要利用指位器做為演算法操作容器元素的媒介。因此，若演算法中必須以隨機方式存取容器內的元素時，被執行演算的容器，其指位器型態就必須支援隨機存取的指位器，否則該演算法將無法執行。

所以，有人以插座與插頭比喻演算法與指位器。演算法就像是一個插座，有的是三孔插座（需要可供向前存取的指位器），有的是兩孔（需要可供隨機存取的指位器）。而指位器這個插頭，有的是三孔（支援向前存取的指位器），有的是兩孔（支援隨機存取的指位器）。此時，每個容器的指位器可以插上哪一種演算法，就可以將該演算法使用於該容器。下圖將利用 sort()、replace()演算法，以及 vector、list 容器說明容器指位器與演算法間的關係。



從上圖可以看出，由於 sort()演算法，需要使用隨機指位器（兩孔插頭），而 vector 容器的指位器即為**隨機指位器**，因此，sort()演算法可作用於 vector 容器。又因為隨機指位器具備向前指位的能力，所以 replace()演算法亦可作用於 vector 容器上（兩孔插頭可以插上三孔插座）。而 list 容器的指位器僅為**向前指位器**，無法滿足 sort()演算法隨機存取的要求，所以 sort()演算法無法使用於 list 容器之上（三孔插頭插不進兩孔插座），但 replace()演算法卻可使用於 list 容器上（兩孔插頭插上兩孔插座剛剛好）。

演算法的使用

STL 的演算法被定義在 algorithm 標頭檔中，運用演算法時，必須載入 algorithm 標頭檔。

```
#include <algorithm>
```

25-5-2 資料編輯演算法

fill()演算法

fill()演算法用於將容器的某個範圍，以特定資料填滿。使用此演算法之容器的指位器至少須為**向前指位器**。

fill(起始指位器, 終止指位器, 欲填滿的資料);

以下敘述將運用『A』字元填滿 v1 容器。

```
fill(v1.begin(), v1.end(), 'A');   //將 v1 用 A 填滿
```

copy()演算法

copy()演算法用於將某一容器的資料複製至某容器內。以下敘述裡，將把 v2 容器的所有元素，複製至 v1 容器第二個元素之後（含第二個元素）。使用此演算法的容器有兩個，被複製之容器（v2）的指位器至少為**向前指位器**，插入資料之容器（v1）的指位器至少為**輸出指位器**。

```
copy(v2.begin(), v2.end(), v1.begin() + 2);
//將 v2 複製到 v1 第二個元素之後
```

remove()演算法

remove()可以將容器某一範圍內，特定內容之元素移除。使用此演算法之容器的指位器至少為向前指位器。

remove(起始指位器, 終止指位器, 欲移除的元素);

remove()移除容器中特定內容之元素時，並不是將該元素從容器中移除，而是將該元素以後的元素向前移動，覆蓋欲移除的元素，且運用 remove()後，容器的長度並不會縮短，而未被移除的元素將會往前複製，所以，remove()將會傳回容器新的末端之指位器，但是後面多餘的元素並不會被刪除。此時，必須再利用 erase()將新尾端至原容器尾端的元素移除。以下敘述將從 v1 容器內，移除內容為『A』的元素，再以 erase()將多餘的元素移除。

```
new_end = remove(v1.begin(), v1.end(), 'A');//將 v1 裡的 A 移除
v1.erase(new_end, v1.end());
```

下圖為以上敘述的執行過程。

原字母	A	A	B	B	B	B	A	A
執行 replace()函數後	B	B	B	B	B	B	A	A
執行 earse()函數後	B	B	B	B				

replace()演算法

replace()用於將容器某範圍內的某資料置換為另一種資料。以下敘述將把 v1 容器內，內容為 B 的元素置換為 C。使用此演算法之容器的指位器至少為向前指位器。

```
replace(v1.begin(), v1.begin()+2, 'B', 'C');
//將 v1 中前兩個 B 置換成 C
```

以下範例將示範這四種演算法的使用。

範例 25-6：fill()、copy()、remove()、replace()演算法

[執行結果]

```
Fill v1 with 'A'...
A A A A A A A A A A
Copy v2 to v1...
A A B B B B A A A A
Remove 'A' from v1...
B B B B
Replace 'B' with 'C'...
C C B B
請按任意鍵繼續 . . .
```

[程式內容]

檔案位置：ex25-6\stl_alg_data.cpp

```
001   /*
002   範例檔名：stl_alg_data.cpp
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007   #include <algorithm>   //載入 algorithm 標頭檔
008   #include <vector>
009
010   using namespace std; //使用 std 名稱空間
011
012   template <class T>
013   void print(vector<T> & pri_con)     //用於列印容器內容的樣版函數
014   {
015      if(pri_con.empty()) //判別 pri_con 是否有資料
016         cout << "Container is empty!" << endl;
017      else{
018         typename vector<T>::iterator i;   //宣告指位器
019
020          //利用 for 迴圈列印 container 元素
021         for(i = pri_con.begin(); i != pri_con.end(); i++)
022         { cout << *i << " "; }   //利用指位器取得 container 內的元素
023         cout << endl;
024      }
```

```
025   }
026
027   int main()  <-- 程式進入點
028   {
029      vector<char> v1(10); //宣告容納 10 個元素的 vector 容器
030
031      cout << "Fill v1 with 'A'..." << endl;
032      fill(v1.begin(), v1.end(), 'A');     //將 v1 用 A 填滿
033      print(v1); //輸出 vector 容器中的元素
034
035      cout << "Copy v2 to v1..." << endl;
036      char a[] = {'B','B','B','B'};
037      vector<char> v2(a, a+4);
038
039      copy(v2.begin(), v2.end(), v1.begin() + 2);
040      //將 v2 複製到 v1 第二個元素之後
041      print(v1);
042
043      cout << "Remove 'A' from v1..." << endl;
044      vector<char>::iterator new_end;
045      new_end = remove(v1.begin(), v1.end(), 'A'); //將 v1 裡的 A 移除
046      v1.erase(new_end, v1.end()); //刪除 v1 中的元素
047      print(v1);
048
049      cout << "Replace 'B' with 'C'..." << endl;
050      replace(v1.begin(), v1.begin()+2, 'B', 'C');
051       //將 v1 中前兩個 B 置換成 C
052      print(v1);
053
054      system("PAUSE");
055      return 0;
056   } //主程式結束
```

25-5-3 搜尋演算法

find()演算法

find()演算法用於尋找容器某範圍內，是否存在某種資料，若容器內找不到該資料，則傳回尋找範圍的結尾指位器。使用此演算法之容器的指位器至少為**輸入指位器**，語法如下：

find(起始指位器, 終止指位器, 欲尋找的資料);

以下敘述將尋找 v1 容器中，是否存在內容為『G』的元素。

```
iter = find(v1.begin(), v1.end(), 'G');//尋找'G'是否在容器中
```

若在 v1 容器中找不到『G』，則 iter 將等於『v1.end()』。

search()演算法

search()演算法用於尋找 A 容器的某範圍中，是否存在 B 容器的某片段，從下面的 search()語法中，可以看出使用 search()時，必須指定兩個容器的範圍。使用此演算法的兩個容器之指位器至少為向前指位器。

search(起始指位器A, 終止指位器A, 起始指位器B, 終止指位器B);

以下敘述將尋找在 v1 中是否存在 v2 容器。

```
iter = search(v1.begin(), v1.end(), v2.begin(), v2.end());
```

以下將示範 find()、search()演算法的使用。

範例 25-7：find()、search()演算法

[執行結果]

```
v1：B G C E G
'G' is in container!
v1：B G C E G
v2：G C
'GC' is in container!
請按任意鍵繼續 . . .
```

[程式碼]

檔案位置：ex25-7\stl_alg_find.cpp

```
001   /*
002   範例檔名：stl_alg_find.cpp
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
```

```
007  #include <algorithm>
008  #include <vector>
009
010  using namespace std; //使用 std 名稱空間
011
012  template <class T>
013  void print(vector<T> & pri_con)    //用於列印容器內容的樣版函數
014  {
015     if(pri_con.empty()) //判別 pri_con 是否有資料
016        cout << "Container is empty!" << endl;
017     else{
018        typename vector<T>::iterator i;   //宣告指位器
019
020         //利用 for 迴圈列印 container 元素
021        for(i = pri_con.begin(); i != pri_con.end(); i++)
022        { cout << *i << " "; }   //利用指位器取得 container 內的元素
023        cout << endl;
024     }
025  }
026
027  int main()   程式進入點
028  {
029     char a[] = {'B','G','C','E','G'};
030     vector<char> v1(a, a+5), v2(a+1, a+3);    //宣告 v1 與 v2 容器
031     vector<char>::iterator iter;    //宣告 vector<char>指位器
032
033     cout << "v1 : ";
034     print(v1);
035
036     iter = find(v1.begin(), v1.end(), 'G');  //尋找'G'是否在容器中
037
038     if(iter == v1.end())   //指位器指向容器最後一個元素，表示不存在
039        cout << "'G' is not in container!" << endl;
040     else
041        cout << "'G' is in container!" << endl;
042
043     cout << "v1 : ";
044     print(v1);
045     cout << "v2 : ";
046     print(v2);
047     iter = search(v1.begin(), v1.end(), v2.begin(), v2.end());
048     //尋找 v1 容器裡是否存在與 v2 容器相同之片段
049
050     if(iter == v1.end())
```

```
051         cout << "'GC' is not in container!" << endl;
052      else
053         cout << "'GC' is in container!" << endl;
054
055       system("PAUSE");
056       return 0;
057    } //主程式結束
```

25-5-4 比對演算法

equal()演算法

equal()用於比對兩容器的某片段是否相等，若兩容器片段相等，equal()將會傳回 true，反之則傳回 false。使用此演算法之容器的指位器至少為輸入指位器，語法如下：

equal(容器 1 的起始指位器, 容器 1 的終止指位器,
容器 2 的起始指位器);

以下敘述將比對整個 v1 與 v2 是否相等。

```
bool result = equal(v1.begin(), v1.end(), v2.begin());
```

mismatch()演算法

mismatch()用於比對兩容器的某範圍內，第一個內容不相同之元素，並傳回此兩不同元素之內容。使用此演算法之容器的指位器至少須為輸入指位器。

mismatch(容器 1 的起始指位器, 容器 1 的終止指位器,
容器 2 的起始指位器);

而 mismatch()的傳回值是一個 pair 物件，裡面包含了兩個指位器，分別為指向容器 1 與容器 2 的元素（有關 pair 的說明請參考 25-4-2 節的範例 25-5）。因此，必須定義一個 pair 物件儲存 mismatch()的回傳值。

```
pair<vector<char>::iterator, vector<char>::iterator> diff_elem;
```

```
//宣告一個由兩個 vector 指位器組成的 pair
diff_elem = mismatch(v1.begin(), v1.end(), v2.begin());
//v1、v2 都是 vector 容器
```

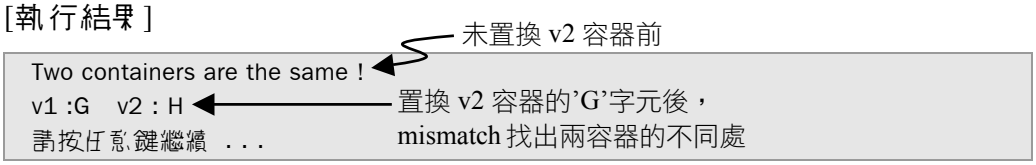
存取 pair 物件內的兩個指位器時，可透過 first 與 second 屬性。

```
diff_elem.first; //使用 pair 的第一個指位器
diff_elem.second; //使用 pair 的第二個指位器
```

以下是 equal()、mismatch()演算法的使用範例。

範例 25-8：equal()、search()演算法

[執行結果]



[程式內容]

檔案位置：ex25-8\stl_alg_com.cpp

```
001  /*
002  範例檔名：stl_alg_com.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <algorithm>
008  #include <vector>
009
010  using namespace std; //使用 std 名稱空間
011
012  int main()  程式進入點
013  {
014     char a[] = {'B','G','C','E'};
015     vector<char> v1(a, a+4), v2(a, a+4); //宣告 v1 容器
016
017     bool result = equal(v1.begin(), v1.end(), v2.begin());
018     //比較 v1 與 v2 是否相同
019
020     if(result)
021        cout << "Two containers are the same !" << endl;
022     else
```

```
023         cout << "Two containers are different !" << endl;
024
025     replace(v2.begin(), v2.end(), 'G', 'H'); //將 v2 中的 G 置換成 H
026
027     pair<vector<char>::iterator, vector<char>::iterator> diff_elem;
028     //宣告一個由兩個 vector 指位器組成的 pair
029     diff_elem = mismatch(v1.begin(), v1.end(), v2.begin());
030      //比對兩者不一樣的地方
031     cout << "v1 :" << *(diff_elem.first) << "  v2 : "
032          << *(diff_elem.second) << endl;
033     //輸出兩字串不同的地方
034
035      system("PAUSE");
036      return 0;
037  } //主程式結束
```

25-5-5 排序相關演算法

sort()演算法

sort()用於排序容器某範圍內的元素，以下敘述將執行 v1 容器內元素的排序，使用此演算法之容器的指位器至少須為**隨機指位器**。

```
    sort(v1.begin(), v1.end());    //排列 v1 內的元素
```

merge()演算法

merge()用於合併兩個容器，並儲存成另一個容器。以下敘述將合併 v1 與 v2，然後儲存至 v3 容器。v1 與 v2 容器的指位器至少為**輸入指位器**，而 v3 容器的指位器則至少為**輸出指位器**。

```
    merge(v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin());
    //合併 v1 與 v2
```

範例 25-9：sort()、merge()演算法

[執行結果]

```
Sorting v1's element...
B C E G
Merge v1 and v2...
B B C E G G C E
請按任意鍵繼續 . . .
```

[程式內容]

檔案位置：ex25-9\stl_alg_sort.cpp

```
001  /*
002  範例檔名：stl_alg_sort.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <algorithm>
008  #include <vector>
009
010  using namespace std;
011
012  template <class T>
013  void print(vector<T> & pri_con)     //用於列印容器內容的樣版函數
014  {
015     if(pri_con.empty()) //判別 pri_con 是否有資料
016        cout << "Container is empty!" << endl;
017     else{
018        typename vector<T>::iterator i;   //宣告指位器
019
020         //利用 for 迴圈列印 container 元素
021        for(i = pri_con.begin(); i != pri_con.end(); i++)
022        { cout << *i << " "; }   //利用指位器取得 container 內的元素
023        cout << endl;
024     }
025  }
026
027  int main()   程式進入點
028  {
029     char a[] = {'B','G','C','E'};
030     vector<char> v1(a, a+4), v2(a, a+4);
```

```
031
032      cout << "Sorting v1's element..." << endl;
033      sort(v1.begin(), v1.end());//排列 v1 內的元素
034      print(v1);  //輸出 vector 容器中的元素
035
036      vector<char> v3(v1.size() + v2.size());
037      //宣告大小可同時容納 v1 與 v2 的容器
038
039      cout << "Merge v1 and v2..." << endl;
040      merge(v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin());
041      //合併 v1 與 v2
042      print(v3);
043
044       system("PAUSE");
045       return 0;
046   } //主程式結束
```

25-5-6 算術演算法

count()演算法

count()演算法用於計算容器中某範圍內某種元素的個數，回傳值為該種元素的個數。使用此演算法之容器的指位器至少為**輸入指位器**。以下敘述將計算 v1 容器中，元素值等於 4 的元素個數。

```
num = count(v1.begin(), v1.end(), 4);   //計算 v1 內'4'的個數
```

max_element()與 min_element()演算法

max_element()與 min_element()演算法分別用於尋找容器中元素的最大值與最小值，並傳回該元素的指位器。使用此演算法之容器的指位器至少為**向前指位器**，語法如下：

max_element(起始指位器, 終止指位器)
min_element(起始指位器, 終止指位器)

for_each()演算法

for_each()是將某一函數套用在容器某範圍的每個元素上，但不更改容器內元素的值。使用此演算法之容器的指位器至少為輸入指位器，語法如下：

for_each(起始指位器, 終止指位器, 函數名稱);

for_each()的前兩個參數，將定義出容器內欲套用函數的元素範圍，而第三個參數則將欲套用函數的名稱傳給 for_each()。以下敘述將把 print()函數套用到 v1 容器的每個元素中。

```
for_each(v1.begin(), v1.end(), print);
//把 print 函數套用在容器每個元素
```

print()函數的定義方式如下所示，回傳值為 void，傳入參數只有一個，且參數的型態應該與套用容器之資料型態相同。（v1 容器為容納 int 型態資料的 vector 容器）。

```
void print(int i) {cout << i << " ";}
```

transform()演算法

transform()與 for_each()類似，只是 transform()會依照函數傳回的計算值，以更改容器內的元素值。使用此演算法之容器的指位器至少為輸入指位器，語法如下：

transform(起始指位器, 終止指位器, 函數名稱);

以下敘述將把 v1 容器內的元素，套用 Square()所定義的方式計算。

```
transform(v1.begin(), v1.end(), v2.begin(), Square);
```

Square()函數的內容如下所示。

```
int Square(int i) { return i * i; }
```

請注意！Square()函數之參數與回傳值的型態，必須與 transform()所作用的容器所容納之資料型態相同。

範例 25-10：算術演算法的使用

[執行結果]

```
Counting number of 'G' in v1...
There are 2 number 4 in v1.
Maxium element : 9
Minium element : 1
Print vector...
1 4 7 2 9 4 3
transform element...
1 16 49 4 81 16 9
請按任意鍵繼續 . . .
```

利用 for_each()套用 print()函數

利用 transform()套用 square()函數

[程式內容]

檔案位置：ex25-10\stl_alg_math.cpp

```
001  /*
002  範例檔名：stl_alg_math.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <algorithm>
008  #include <vector>
009
010  using namespace std; //使用 std 名稱空間
011
012  //將由 for_each 與 transform 套用的函數
013  int Square(int i) { return i * i; }
014  void print(int i) {cout << i << " ";}
015
016  int main()
017  {
018     int a[] = {1,4,7,2,9,4,3};
019     vector<int> v1(a, a+7);
020     int num = 0;
021
022     cout << "Counting number of 'G' in v1..." << endl;
023     num = count(v1.begin(), v1.end(), 4);     //計算 v1 內'G'的個數
024     cout << "There are " << num << " number 4 in v1." << endl;
025
026     cout << "Maxium element : "
027          << *( max_element(v1.begin(), v1.end())) << endl;
```

程式進入點

```
028      //輸出容器中最大的元素
029
030      cout << "Minium element : "
031           << *( min_element(v1.begin(), v1.end())) << endl;
032      //輸出容器中最小的元素
033
034      cout << "Print vector..." << endl;
035      for_each(v1.begin(), v1.end(), print);
036      //將 print 套用在容器每個元素
037
038      cout << endl;
039
040      cout << "transform element..." << endl;
041      vector<int> v2(v1.size());
042      transform(v1.begin(), v1.end(), v2.begin(), Square);
043      //將容器的每個元素利用 Square 轉換成其平方值
044      for_each(v2.begin(), v2.end(), print);
045      cout << endl;
046
047       system("PAUSE");
048       return 0;
049   } //主程式結束
```

25-6 函數物件 專家

25-6-1 函數物件的應用

函數物件與關聯容器

在許多演算法裡，常常會用到大小比較、邏輯運算以及算術運算...等演算邏輯。在 STL 裡，提供了許多已經完成撰寫的函數物件，供程式設計師運用。還記得『less<>』嗎？在 25-4 節介紹關聯容器時，曾經使用過，它就是一個函數物件。

當定義關聯容器時，透過『less<>』的使用，關聯容器內元素的索引值將以**遞增**方式排列。若希望容器以遞減方式排列，當然也可以，只需要將『greater<>』函數物件取代為『less<>』函數物件即可。這樣容器內的索引值，便將以**遞減**方式排列。您可以試著將 25-4 節的**範例 25-4** 與**範例 25-5** 中，宣告容器時所用的『less<>』函數物件改成『greater<>』函數物件，看看執行結果有什麼不同。

```
map<string, string, greater <string> >  m;
//宣告 map 容器，關鍵值以遞減排列
```

使用的標頭檔

當欲使用函數物件時，必須載入 functional 標頭檔。

```
#include <functional>
```

函數物件與演算法

函數物件的應用還不止如此，記得 sort()演算法嗎？sort()預設的排序方式也是遞增，我們一樣可以利用函數物件將 sort()的排序方式改為遞減。您可以修改 25-5-5 節的**範例 25-9**。首先，必須將定義函數物件的 functional 標頭檔載入。

```
#include <functional>
```

然後修改第 33 行，呼叫 sort()演算法的參數，原程式碼如下：

```
033   sort(v1.begin(), v1.end());    //排列 v1 內的元素
```

將函數物件傳入 sort()中，更改如下所示。

```
033   sort(v1.begin(), v1.end(), greater<char>());//排列 v1 內的元素
```

由於函數物件也是樣版類別，所以在使用函數物件時，必須傳入資料型態，在這裡必須注意！函數物件使用的型態，與容器所容納的資料型態必須**相同**。**範例 25-9** 裡 v1 是一個容納 char 的 vector 容器，因此，使用 greater 函數物件時，所套用的資料型態也必須是 char。

到底函數物件是什麼

函數物件是一個由樣版類別所產生的物件，而產生函數物件的樣版類別什麼都沒有，只有一個用於過載的 operator()函數。所以，25-4-2 節的**範例 25-5** 中，若希望利用 greater 函數物件做為關聯容器排序依據時，必須以『greater<string>』做為第三個樣版參數（greater<string>代表 greater 函數物件的樣版類別將套用 string 型態）。

```
map<string, string, greater <string> > m;
//宣告 map 容器，索引值以遞減排列
```

當將函數物件傳遞給演算法時，必須先呼叫樣版類別的預設建構子，建立一個函數物件，然後才能夠傳遞給容器或演算法使用。所以，欲更改 25-5-5 節**範例 25-9** 的第 33 行時，sort()的第三個參數必須要用『greater<char>()』呼叫樣版類別的預設建構子，以產生函數物件，然後傳入 sort()演算法。

```
033   sort(v1.begin(), v1.end(), greater<char>());//排列 v1 內的元素
```

25-6-2 STL 提供的函數物件

STL 提供了三類函數物件供程式設計師使用，這三類函數物件的名稱與功能說明如下：

函數物件	基本功能說明
算術	
plus<T>	將型態為 T 的兩參數相加，並傳回相加的和。
minus<T>	將型態為 T 的兩參數相減，並傳回相減的差。
times<T>	將型態為 T 的兩參數相乘，並傳回相乘的積。
divides<T>	將型態為 T 的兩參數相除，並傳回相除的商。
modulus<T>	將型態為 T 的兩參數相除取餘數，並傳回餘數。
negate<T>	傳回型態為 T 的參數的相反值。

函數物件	基本功能說明
比較	
equal_to<T>	接受兩型態為 T 的參數（x、y），若 x==y 傳回 true，否則傳回 false。
not_equal_to<T>	接受兩型態為 T 的參數（x、y），若 x!=y 傳回 true，否則傳回 false。
greater	接受兩型態為 T 的參數（x、y），若 x>y，則傳回 true，否則傳回 false。
less	接受兩型態為 T 的參數（x、y），若 x<y，則傳回 true，否則傳回 false。
greater_equal	接受兩型態為 T 的參數（x、y），若 x>=y，則傳回 true，否則傳回 false。
less_qual	接受兩型態為 T 的參數（x、y），若 x<=y，則傳回 true，否則傳回 false。
邏輯	
logical_and	接受兩型態為 T 的參數（x、y），並傳回 x&&y 後的布林值。
logical_or	接受兩型態為 T 的參數（x、y），並傳回 x||y 後的布林值。
logical_not	接受型態為 T 的參數 x，並傳回 !x 的布林值。

以下範例，將擇要示範兩種函數物件與演算法的配合使用。

範例 25-11：函數物件的使用

[執行結果]

```
transform element...
v1 : 1 4 7 2 9 4 3
v2 : 1 4 7 2 9 4 3
v3 (v1 + v2) :2 8 14 4 18 8 6
sort element..
v3 : 18 14 8 8 6 4 2
請按任意鍵繼續 . . .
```

1.用 transform 相加

2.利用 sort 排序

[程式內容]

檔案位置：ex25-11\stl_fun_math.cpp

```
001  /*
002  範例檔名：stl_fun_math.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <functional>  //載入 functional 標頭檔
008  #include <algorithm>
009  #include <vector>
010
011  using namespace std; //使用 std 名稱空間
012
013  //將由 for_each 套用的函數
014  void print(int i) {cout << i << " ";}
015
016  int main()   程式進入點
017  {
018     int a[] = {1,4,7,2,9,4,3};
019     vector<int> v1(a, a+7), v2 = v1, v3(v1.size());
020
021     cout << "transform element..." << endl << "v1 : ";
022     for_each(v1.begin(), v1.end(), print);
023     cout << endl << "v2 : ";
024     for_each(v2.begin(), v2.end(), print);
025     cout << endl << "v3 (v1 + v2) :";
026     transform(v1.begin(), v1.end(), v2.begin(),
                                      v3.begin(), plus<int>());
027     //將 v1 與 v2 的元素相加，並儲存至 v3
028
029     for_each(v3.begin(), v3.end(), print);
030     cout << endl;
031
032     cout << "sort element.." << endl << "v3 : ";
033     sort(v3.begin(), v3.end(), greater<int>());
034     //以遞減方式排列容器內元素
035
036     for_each(v3.begin(), v3.end(), print);
037     cout << endl;
038
039     system("PAUSE");
040     return 0;
```

```
041    } //主程式結束
```

[程式說明]

第 26 行：在 25-5-6 節的**範例 25-10** 裡，利用 transform()演算法將 v1 元素的值，透過 Square()函數的運算，轉換至 v2。在這個範例將示範 transform()的另一種用法，將 v1 與 v2 利用『plus<>』相加，然後儲存至 v3。請注意！v1、v2、v3 與『plus<>』套用的型態必須相同。

第 33 行：運用『greater<int>』函數物件，將 v3 容器的元素以遞減方式排列。

25-6-3　自訂函數物件

雖然 STL 提供許多函數物件時，但是總有無法滿足使用者需求的時候，此時，可以透過自行建立的函數物件，滿足較為特殊的計算需求。25-5-6 節的**範例 25-10** 裡，transform()演算法將透過自訂的 Square()函數，將 v1 容器內元素平方，然後儲存至 v2 容器。

```
'摘自範例 25-10
042    transform(v1.begin(),v1.end(),v2.begin(),Square);
```

這種方式並不是傳遞一個物件函數，而只是傳遞一個函數（正確的說應該是函數指標），這兩種方式的不同處將於稍後說明。25-6-1 節曾提到函數物件裡，只有一個過載『()』運算子的 operator()樣版函數。以下敘述將把 Square 類別，定義為函數物件類別（改寫 25-5-6 節**範例 25-10** 的 Square()函數），由此您可看出函數物件的最基本型態。

```
class Square //函數物件類別
{
public:
    //定義過載的()運算子做爲函數物件的樣版函數
   template <class T>
    T operator() (T i) { return i * i; }
};
```

在演算法中傳遞函數物件的方式如下：

```
transform(v1.begin(), v1.end(), v2.begin(), Square());
```

有別於傳遞函數，傳遞函數物件時，將會呼叫函數物件類別的預設建構子，建立一個函數物件。在上述的 Square 函數物件類別裡，預設建構子將由編譯器自動產生。在 functional 標頭檔裡，將函數物件定義為樣版類別的形式，以下敘述將把以上敘述內的 Square 類別，改寫成樣版類別。

```
template <class T>  //樣版函數物件類別
class tmpSquare
{
public:
   T operator() (T i) { return i * i; }
};
```

改寫成樣版類別的 Square 類別，呼叫方式與由 STL 所提供的相同，必須傳遞與演算法操作容器相同的型態。以下敘述內，v1、v3 所容納的資料型態為 int。

```
transform(v1.begin(), v1.end(), v3.begin(), tmpSquare<int>());
```

以下為自訂函數物件的完整範例。

範例 25-12：自訂函數物件

[執行結果]

```
transform element by function object...
v2 : 1 16 49 4 81 16 9
transform element by template function object...
v3 : 1 16 49 4 81 16 9
請按任意鍵繼續 . . .
```

[程式內容]

檔案位置：ex25-12\stl_fun_def.cpp

```
001   /*
002   範例檔名：stl_fun_def.cpp
003   程式開發：郭尚君
004   */
```

```
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <algorithm>
008  #include <vector>
009
010  using namespace std; //使用 std 名稱空間
011
012  class Square //函數物件類別
013  {
014  public:
015      //定義做爲函數物件的樣版函數
016     template <class T>
017      T operator() (T i) { return i * i; }
018  };
019
020  template <class T>  //樣版函數物件類別
021  class tmpSquare
022  {
023  public:
024     T operator() (T i) { return i * i; }
025  };
026
027  void print(int i) {cout << i << " ";}
028
029  int main()  <-- 程式進入點
030  {
031     int a[] = {1,4,7,2,9,4,3};
032     vector<int> v1(a, a+7), v2(v1.size()), v3(v1.size());
033
034     cout << "transform element by function object..."
035          << endl <<"v2 : ";
036     transform(v1.begin(), v1.end(), v2.begin(), Square());
037     //利用函數物件將元素轉換成平方值
038
039     for_each(v2.begin(), v2.end(), print);
040     cout << endl << "transform element by template function object...";
041     cout << endl << "v3 : ";
042
043     transform(v1.begin(), v1.end(), v3.begin(), tmpSquare<int>());
044     //利用由樣版函數物件將元素轉換成平方值
045
046     for_each(v3.begin(), v3.end(), print);
047     cout << endl;
048
```

```
049       system("PAUSE");
050       return 0;
051    } //主程式結束
```

25-7 轉接器 專家

25-7-1 簡介轉接器

容器轉接器（Container Adaptor）

容器轉接器用於將循序容器轉換成另一種容器，也就是以循序容器為基礎，修正得到新的容器。以 stack 轉接器為例，stack 的特性為後進先出（LIFO，Last In First Out），用於產生 stack 的容器必須要具備 push_back()、pop_back() 的能力，而循序容器中 vector、list、deque 均符合這個條件，所以它們都可以透過轉接器，將其轉接成 stack 容器。由於運用轉接器產生的並不是真正的容器，因此無法使用指位器。各種容器轉接器的名稱與特性說明如下：

轉接器名稱	可被轉換的容器	特性
stack	以 vector、list、deque 轉接而成，預設為 deque。	只從一端插入（push）、移除（pop）元素，遵循後進先出（LIFO，Last In First Out）規則。
queue	以 list、deque 轉接而成，預設為 deque。	只從一端插入（push）、移除（pop）元素，遵循先進先出（FIFO，First In First Out）規則。
priority_queue	以 vector、deque 轉接而成，預設為 vector。	容器內各元素按照指定順序排列。插入時，直接按插入到適當的位置。移除時，從最頂端開始移除。

25-7-2 stack 轉接器

stack 轉接器內，元素所遵守的原則為後進先出（LIFO，Last In First Out）。因此，要轉換為 stack 的序列容器必須具備 push_back()、pop_back()、size()、empty()這些成員函數。在序列容器中，每一種容器都具備些特性，所以，每個序列容器都可以被轉換為 stack。若未指定轉接器使用的序列容器為何時，預設使用的序列容器為 deque。

使用 stack 轉接器

使用 stack 須載入 stack 標頭檔

```
#include <stack>
using namespace std;
```

宣告一個 stack 轉接器的語法如下：

這裡要插入空白

```
stack <資料型態, 轉換的容器<資料型態> > 變數名稱;
```

以下敘述將宣告 List_Stack 物件為利用 list 建立容納 int 型態資料的 stack 容器。

```
stack <int, list<int> > List_Stack;     //利用 list 容器建立 stack
```

這裡要插入空白

stack 轉接器的方法

下表將列出 stack 轉接器的成員函數。

名稱	用途
插入與刪除	
push	增加容器元素
pop	刪除容器元素
取出容器	
top	傳回容器頂端元素的參考
size	傳回目前容器中的元素個數
empty	若容器內目前無元素傳回 true，反之傳回 false

範例 25-13：stack 的使用範例

[執行結果]

```
Pop from Deque_Stack
3 2 1 0
Pop from Vec_Stack
3 2 1 0
Pop from List_Stack
3 2 1 0
請按任意鍵繼續 . . .
```

[程式內容]

檔案位置：ex25-13\stl_stack.cpp

```
001  /*
002  範例檔名：stl_stack.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <stack> //載入 stack 標頭檔
008  #include <vector>
009  #include <list>
010
011  using namespace std; //使用 std 名稱空間
012
013  template <class T> //輸出 stack 元素的樣版函數
014  void printStack(T & s)
015  {
016     while(!s.empty()) //判別容器是否爲空的
017     {
018        cout << s.top() << " "; //輸出 stack 頂端的元素資料
019        s.pop(); //移除 stack 頂端的元素資料
020     }
021     cout << endl;
022  }
023
024  int main() //主程式開始  程式進入點
025  {
026     stack<int> Deq_Stack;  //利用預設的 deque 容器建立 stack
027     stack<int, vector <int> > Vec_Stack; //利用 vector 容器建立 stack
028     stack <int, list<int> > List_Stack;  //利用 list 容器建立 stack
```

```
029
030      for (int i = 0; i < 4; i++)
031      {
032         Deq_Stack.push(i);  //將元素利用 push 新增至容器
033         Vec_Stack.push(i);
034         List_Stack.push(i);
035      }
036
037      cout << "Pop from Deque_Stack" << endl;
038      printStack(Deq_Stack); //輸出容器內的元素
039      cout << "Pop from Vec_Stack" << endl;
040      printStack(Vec_Stack);
041      cout << "Pop from List_Stack" << endl;
042      printStack(List_Stack);
043
044      system("PAUSE");
045      return 0;
046   } //主程式結束
```

[程式說明]

第 13～22 行：在輸出 stack 元素的 printStack()函數內，利用 while 迴圈，以及 stack 的 top()、pop()成員函數輸出 stack 內元素的資料。

第 27 行：宣告 stack 轉接器時，第二個樣版參數將用於指定 stack 所使用的容器類別，請注意！最後一個『>』與倒數第二個『>』中間必須插入空白，否則將會被編譯器誤認為『>>』運算子。

第 30～35 行：利用 for 迴圈將資料（0、1、2、3）插入到容器中，從執行結果可看出 stack 轉接器的元素，取得容器內元素之規則的確為**後進先出**。

25-7-3 queue 轉接器

取得 queue 轉接器內元素所遵守的原則為**先進先出**（FIFO，First In First Out）。因此，要轉換為 queue 的序列容器必須具備 push_back()、pop_front()、back()、size()、empty()這些方法。在序列容器中，除了 vector 容器外，deque 與 list 都具備些特性，所以，deque 與 list 都可以被轉換為 queue，預設使用的容器為 deque。

使用 queue 轉接器

使用 queue 須載入 queue 標頭檔

```
#include <queue>
using namespace std;
```

宣告一個 queue 轉接器的語法如下：

這裡要插入空白

```
queue <資料型態, 轉換的容器<資料型態> > 變數名稱;
```

以下敘述將宣告 List_queue 為利用 list 所建立，容納 int 型態資料的 queue 轉接器。

```
queue <int, list<int> > List_ queue; //利用 list 容器建立 queue
```

這裡要插入空白

queue 轉接器的方法

下表中將列出 queue 轉接器的成員函數。

名稱	用途
插入與刪除	
push	增加容器元素
pop	增加容器元素
取出容器	
front	傳回容器前端元素的參考
back	傳回容器末端元素的參考
size	傳回目前容器中的元素個數
empty	若容器內目前無元素傳回 true，反之傳回 false

範例 25-14：queue 的使用

[執行結果]

```
Pop from Deq_Queue
0 1 2 3
```

```
Pop from List  Queue
0 1 2 3
請按任意鍵繼續 . . .
```

[程式執行結果]

檔案位置：ex25-14\stl_queue.cpp

```
001  /*
002  範例檔名：stl_queue.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstring> //在 std 名稱空間內載入 C 語言的 string.h
007  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
008  #include <queue> //載入 queue 標頭檔
009  #include <list>
010
011  using namespace std; //使用 std 名稱空間
012
013  template <class T>  //輸出容器元素的樣版函數
014  void printQueue(T & q)
015  {
016     while(!q.empty())    //判別容器是否爲空的
017     {
018        cout << q.front() << " ";     //取得容器頂端元素的參考
019        q.pop(); //從容器前端移除元素
020     }
021     cout << endl;
022  }
023
024  int main() //主程式開始  程式進入點
025  {
026     queue<int> Deq_Queue;  //以預設容器建立 queue
027     queue< int, list<int> > List_Queue;  //以 list 建立 queue
028
029     for (int i = 0; i < 4; i++)
030     {
031        Deq_Queue.push(i);  //增加元素至容器中
032        List_Queue.push(i);
033     }
034
035     cout << "Pop from Deq_Queue" << endl;
036     printQueue(Deq_Queue); //列印容器內的元素
```

```
037      cout << "Pop from List_Queue" << endl;
038      printQueue(List_Queue);
039
040      system("PAUSE");
041      return 0;
042   } //主程式結束
```

[程式說明]

第 13～22 行：在輸出 queue 元素的 printQueue()函數內，將利用 while 迴圈，以及 queue 的 front()、pop()成員函數輸出 queue 容器內元素的資料。

第 29～33 行：利用 for 迴圈依序將 0、1、2、3 增加至容器內。

第 36、38 行：呼叫 printQueue()樣版函數，輸出容器內的元素。從執行結果，可以看出容器內的元素將按照先進先出原則輸出。

25-7-4 priority_queue 轉接器

priority_queue 與 queue 功能大致是差不多的，只是 priority_queue 會執行容器內元素的排序。

使用 priority_queue 轉接器

使用 priority_queue 時，必須載入 queue 及 functional 標頭檔。

```
#include <deque>
#include <functional>
using namespace std;
```

宣告一個 priority_queue 轉接器的語法如下：

這裡要插入空白

priority_queue <資料型態, 轉換的容器<資料型態> > 變數名稱;

以下敘述將宣告 Deq_PriQueue 物件為利用 deque 容器，所建立容納 int 型態資料的 priority_queue 轉接器。

這裡要插入空白

```
priority_queue <int, deque<int> > Deq_PriQueue;
//利用 deque 容器建立 priority_queue
```

priority_queue 轉接器的方法

下表將列出 priority_queue 轉接器的成員函數。

名稱	用途
插入與刪除	
push	增加容器元素
pop	增加容器元素
取出容器	
front	傳回容器前端元素的參考
back	傳回容器末端元素的參考
size	傳回目前容器中的元素個數
empty	若容器內目前無元素傳回 true，否則傳回 false

範例 25-15：priority_queue 的使用

[執行結果]

```
Pop from Vec_PriQueue
3 3 2 2 1 1 0
Pop from Deq_PriQueue
3 3 2 2 1 1 0
請按任意鍵繼續 . . .
```

用 vector 轉換而成的 priority_queue

用 deque 轉換而成的 priority_queue

[程式內容]

檔案位置：ex25-15\stl_pri_queue.cpp

```
001   /*
002   範例檔名：stl_pri_queue.cpp
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007   #include <queue> //載入 deque 標頭檔
```

```
008   #include <deque>
009   #include <functional>  //載入 functional 標頭檔
010
011   using namespace std; //使用 std 名稱空間
012
013   template <class T>
014   void printPriQueue(T & p)
015   {
016      while(!p.empty()) //檢查容器是否是空的
017      {
018         cout << p.top() << " "; //輸出容器頂端元素的參考
019         p.pop(); //移除容器頂端元素的參考
020      }
021      cout << endl;
022   }
023
024   int main() //主程式開始   程式進入點
025   {
026      priority_queue<int> Vec_PriQueue;
027      //以預設的 vector 容器建立 priority_queue
028      priority_queue< int, deque<int> > Deq_PriQueue;
029      //以 deque 容器建立 priority_queue
030
031      int i;
032
033      for (i = 0; i < 4; i++)
034      {
035         Vec_PriQueue.push(i);    //以 push 新增元素
036         Deq_PriQueue.push(i);
037      }
038
039      for (i = 3; i > 0; i--)
040      {
041         Vec_PriQueue.push(i);
042         Deq_PriQueue.push(i);
043      }
044
045      cout << "Pop from Vec_PriQueue" << endl;
046      printPriQueue(Vec_PriQueue);    //輸出容器的資料
047      cout << "Pop from Deq_PriQueue" << endl;
048      printPriQueue(Deq_PriQueue);
049
050      system("PAUSE");
051      return 0;
052   } //主程式結束
```

[程式說明]

第 33～43 行：以 0、1、2、3、3、2、1 的順序插入資料，從執行結果可以看出容器內的元素，將以由大到小的方式排列。

25-7-5 倒轉指位器

介紹序列容器時，常常會看到利用 for 迴圈，以遞增指位器的方式輸出容器內的元素。或許您會希望能夠以相反方向輸出容器裡的元素，這時您或許會寫出這樣的 for 迴圈。

```
for(i = pri_con.end() ; i != pri_con.begin(); i--)
{
   cout << *i << " ";
}
```

但是輸出結果，恐怕不是您想像的那樣。當欲反向輸出陣列元素時，必須使用倒轉指位器，宣告語法如下：

```
容器名稱::reverse_iterator i;  //宣告 i 為倒轉指位器
```

並配合容器的 rbegin()與 rend()成員函數，才能正確地反向輸出容器內的元素。以下敘述將宣告 vector 容器所使用的倒轉指位器，並反向輸出容器內元素的內容。

```
vector::reverse_iterator i;    //宣告 i 爲 vector 的倒轉指位器

//利用 for 迴圈反向列印元素
for(i = pri_con.rbegin() ; i != pri_con.rend(); i++)
{
   cout << *i << " ";  //利用指位器輸出元素
}
```

範例 25-16：倒轉指位器的使用

[執行結果]

```
Printing element :
1 6 4
```

```
Printing element by reverse direction :
4 6 1
請按任意鍵繼續 . . .
```

[程式執行結果]

檔案位置：ex25-16\stl_rev_iter.cpp

```
001  /*
002  範例檔名：stl_rev_iter.cpp
003  程式開發：郭尚君
004  */
005  #include <iostream>
006  #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007  #include <vector> //載入 vector 標頭檔
008
009  using namespace std; //使用 std 名稱空間
010
011  #define Size 4
012
013  template <class T>
014  void print(vector<T> & pri_con)     //用於列印容器內容的樣版函數
015  {
016     if(pri_con.empty()) //判別 pri_con 是否有資料
017        cout << "Container is empty!" << endl;
018     else{
019        typename vector<T>::iterator i;   //宣告指位器
020
021         //利用 for 迴圈列印 container 元素
022        for(i = pri_con.begin() ; i != pri_con.end(); i++)
023        {
024           cout << *i << " ";     //利用指位器取得 container 內的元素
025        }
026        cout << endl;
027     }
028  }
029
030  template <class T>
031  void rev_print(vector<T> & pri_con)      //用於列印容器內容的樣版函數
032  {
033     if(pri_con.empty()) //判別 pri_con 是否有資料
034        cout << "Container is empty!" << endl;
035     else{
036        typename vector<T>::reverse_iterator i; //宣告倒轉指位器
```

```
037
038          //利用 for 迴圈反向列印元素
039         for(i = pri_con.rbegin() ; i != pri_con.rend(); i++)
040         {
041            cout << *i << " "; //利用指位器輸出元素
042         }
043         cout << endl;
044      }
045   }
046
047   int main() //主程式開始  程式進入點
048   {
049      vector<int> v; //宣告儲存整數用的 vector
050
051      v.push_back(1); //從 vector 後端增加元素
052      v.push_back(6);
053      v.push_back(4);
054      cout << "Printing element : " << endl;
055      print(v);   //輸出容器內的資料
056      cout << "Printing element by reverse direction : " << endl;
057      rev_print(v);
058
059      system("PAUSE");
060      return 0;
061   } //主程式結束
```

[程式說明]

第 30～45 行：定義反向輸出容器內元素的 rev_print()樣版函數。在函數內，將宣告用於處理 vector 容器的倒轉指位器（第 36 行），並配合容器的 rbegin()與 rend()成員函數，反向輸出容器裡元素的內容。

25-7-6 插入指位器

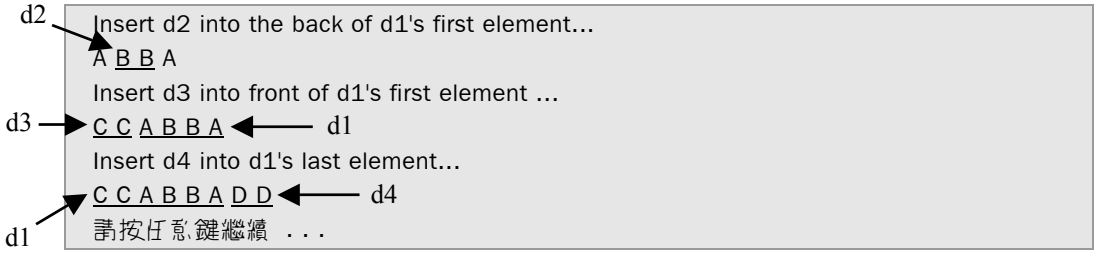
當使用插入指位器時，將可強迫演算法使用容器新增資料的成員函數，如：push_front()、push_back()或者 insert()，欲新增至容器內的資料，將插入指位器所指向的元素。插入指位器有三種，將分別呼叫前述的三種成員函數，列表說明如下：

插入指位器	被使用的成員函數	說明	可使用的容器
front_insert	push_front	將資料插入容器的前端	deque、list
back_insert	push_back	將資料插入容器的末端	所有容器
inserter	insert	將資料插入容器的指定的位置	deque、vector

以下範例將利用 copy()演算法說明三種指位器的使用。

範例 25-17：插入指位器的使用

[執行結果]



[程式內容]

檔案位置：ex25-17\stl_ins_iter.cpp

```
001   /*
002   範例檔名：stl_ins_iter.cpp
003   程式開發：郭尚君
004   */
005   #include <iostream>
006   #include <cstdlib> //在 std 名稱空間內載入 C 語言的 stdlib.h
007   #include <algorithm> //載入 algorithm 標頭檔
008   #include <deque>
009
010   using namespace std; //使用 std 名稱空間
011
012   template <class T>
013   void print(deque<T> & pri_con)//用於列印容器內容的樣版函數
014   {
015      if(pri_con.empty()) //判別 pri_con 是否有資料
016         cout << "Container is empty!" << endl;
017      else{
018         typename deque<T>::iterator i; //宣告指位器
019
```

```
020           //利用 for 迴圈列印 container 元素
021         for(i = pri_con.begin(); i != pri_con.end(); i++)
022         { cout << *i << " "; }   //利用指位器取得 container 內的元素
023         cout << endl;
024      }
025   }
026
027   int main()   程式進入點
028   {
029      deque<char> d1(2, 'A'), d2(2, 'B'), d3(2,'C'), d4(2,'D');
030      //宣告四個容納兩個元素的 deque 容器，並分別以字母填滿
031      cout << "Insert d2 into the back of d1's first element..." << endl;
032      copy(d2.begin(), d2.end(), inserter(d1, d1.begin()+1));
033      //將 d2 插入第一個元素之後
034
035      print(d1);
036
037      cout << "Insert d3 into front of d1's first element ..." << endl;
038      copy(d3.begin(), d3.end(), front_inserter(d1));
039      //將 d2 插入第一個元素之前
040
041      print(d1);
042
043      cout << "Insert d4 into d1's last element..." << endl;
044      copy(d4.begin(), d4.end(), back_inserter(d1));
045      //將 d2 插入最後一個元素之後
046
047      print(d1);
048
049      system("PAUSE");
050      return 0;
051   } //主程式結束
```

[程式說明]

第 32 行：利用 inserter()指位器將 d2 的元素插入到 d1 的第二個元素後，inserter()的第一個參數為欲插入元素的容器，第二個參數是欲插入元素的指位器。

```
032   copy(d2.begin(), d2.end(), inserter(d1, d1.begin()+1));
```

第 38 行：利用 front_inserter()將 d3 的元素插入 d1 的第一個元素之前。front_inserter()之參數為欲插入元素的容器。

```
038       copy(d3.begin(), d3.end(), front_inserter(d1));
```

第 44 行：利用 back_inserter()將 d4 的元素插入 d1 的第一個元素之後。back_inserter()的參數是欲插入元素的容器。

```
044       copy(d4.begin(), d4.end(), back_inserter(d1));
```