

附錄 A 標準樣板函式庫(Standard Template Library)

標準樣板函式庫(Standard Template Library，縮寫為 STL)是以樣板(Template)概念，定義了重複可利用的元件，元件提供儲存資料的容器與對資料進行處理的演算法，可以省去撰寫資料結構與演算法所需的複雜程式，但仍須了解資料結構與演算法的邏輯概念與運作，才能有效利用這些標準樣板函式庫(資料結構與演算法)。

A-1 簡介標準樣板函式庫

標準樣板函式庫分成容器(container)、迭代器(iterator)與演算法(algorithm)三大部分。容器分成循序式容器(sequence containers)、關聯式容器(associative containers)與配接器(container adapters)，循序式容器是一種線性的資料儲存容器，分成 **vector**、**deque** 與 **list** 等三種；關聯式容器為非線性儲存容器，可以快速搜尋資料，可以用於儲存資料的集合，與鍵值與儲存值(key 與 value)配對的資料結構，分成 **set**、**multiset**、**map** 與 **multimap** 等四種；配接器為序列式容器的變形，限制插入與取出資料是在序列式容器的第一個元素或最後一個元素，分成 **stack**、**queue** 與 **priority_queue** 等三種。

迭代器(iterator)類似指標的功能，可以指定容器中個別的元素，可以與演算法功能一起使用。只有 **vector** 與 **deque** 支援隨機的迭代器，而 **list**、**set**、**multiset**、**map** 與 **multimap** 支援雙向的迭代器，但配接器 **stack**、**queue** 與 **priority_queue** 不支援迭代器的使用。

演算法(algorithm)提供搜尋、排序與比較等功能，可以將這些功能應用在不同的容器上，讓演算法與資料容器分開，演算法也可以重複使用，增加演算法的使用彈性，減少使用者撰寫程式的困難與複雜度。

標準樣板函式庫在使用之前，需要包含標頭檔，所需標頭檔與所提供的標準樣板函式庫(Standard Template Library)類別對應如下表。

標頭檔	所提供的標準樣板函式庫(STL)類別
#include <vector>	vector
#include <list>	list
#include <deque>	deque
#include <set>	set 與 multiset
#include <map>	map 與 multimap
#include <stack>	stack
#include <queue>	queue 與 priority_queue
#include <algorithm>	find,sort,count,lower_bound,upper_bound 等

A-2 循序式容器(sequence containers)

循序式容器是一種線性的資料儲存容器，分成 **vector**、**deque** 與 **list** 三種，以下分別介紹這三種容器。

A-2-1 vector

vector 以陣列為儲存資料的容器，系統會動態增加或減少陣列的大小，不須事先宣告 **vector** 的大小，可以利用索引值直接存取指定的資料，在 **vector** 中間插入與刪除資料的成本較 **list** 高，需要移動插入元素所在位置之後的元素，與移動刪除元素所在位置之後的元素。

vector 所提供的重要函式

函式分類	函式	功能	平均效率
迭代器函式	begin()	回傳 vector 的第一個元素	O(1)
	end()	回傳 vector 的最後一個元素的下一個，這個元素不存在，通常用於表示資料已經到達結尾	O(1)
新增或刪除元素函式	push_back(x)	將 x 加到 vector 最後一個元素。	O(1)，若超過 vector 預定的大小，需要調整儲存空間時效率為 O(n)
	pop_back()	刪除最後一個元素。	O(1)
	insert(pos,x)	在 vector 的 pos 位置插入元素 x。	O(n)
	erase(pos)	刪除 vector 的 pos 位置的元素。	O(n)
	clear()	刪除 vector 中所有元素	O(n)
讀取元素函式	at(i) 或 [i]	讀取 vector 第 i 個元素， at(i) 在 i 值超出邊界時會發出錯誤訊息，而 [i] 在 i 值超出邊界時不會發出錯誤訊息。	O(1)
	front()	讀取 vector 第一個元素	O(1)
	back()	讀取 vector 最後一個元素	O(1)
儲存空間函式	empty()	檢查 vector 是否是空的，若是空的，回傳 true，否則回傳 false。	O(1)
	size()	回傳 vector 目前儲存元素的個數	O(1)

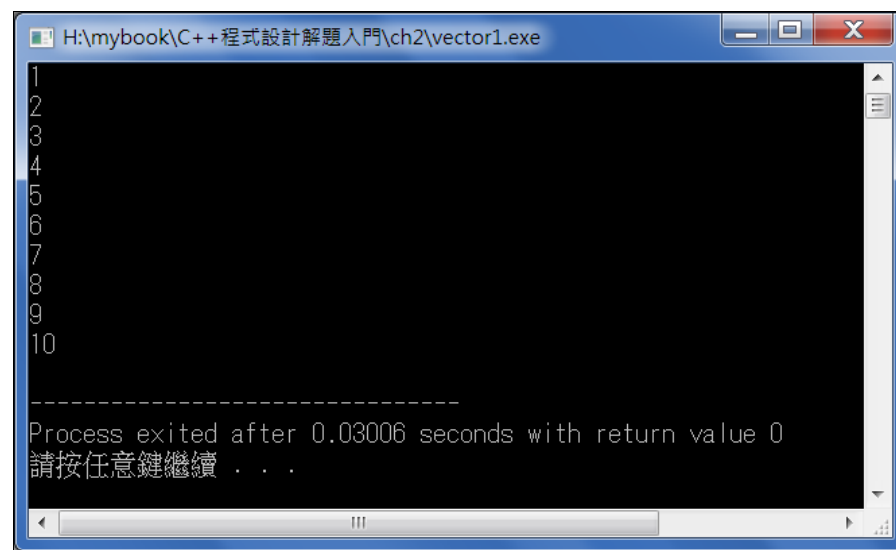
vector 程式範例

A-2-1 在 vector 中新增與讀取元素(chA\A-2-1-vector.cpp)

範例說明

請實作一個程式將數字 1 到 10 依序加入 vector，加入完成後顯示 vector 所有元素值到螢幕。

預期程式執行結果



```
1
2
3
4
5
6
7
8
9
10
-----
Process exited after 0.03006 seconds with return value 0
請按任意鍵繼續 . . .
```

範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 vec 為儲存整數(int)的 vector。 第 6 行：宣告整數 s 並初始化為 10。 第 7 行到第 9 行：使用迴圈在 vec 中依序插入 1、2、3、...、10 到 vec 的最後。 第 10 行到第 12 行：使用迴圈在依序 vec 的所有元素，迴圈變數 i 由 0 到 vec.size()-1，顯示 vec[i] 到螢幕上。
2	#include <vector>	
3	using namespace std;	
4	int main(){	
5	vector<int> vec;	
6	int s=10;	
7	for(int i=1; i<=s;i++) {	
8	vec.push_back(i);	
9	}	
10	for(int i=0;i<vec.size();i++){	
11	cout << vec[i] << endl;	
12	}	
13	}	

A-2-2 deque

以陣列為儲存資料的容器，可以動態增加或減少陣列的大小，兩個端點都可以新增與刪除元素，可以利用索引值直接存取指定的資料，與 **vector** 一樣在中間位置插入與刪除資料的成本較 **list** 高。**deque** 與 **vector** 不同之處在於 **deque** 兩頭都可以新增與刪除元素，而 **vector** 只能由後面新增與刪除元素，且 **deque** 不佔有連續的儲存空間，**vector** 佔有連續的儲存空間，當新增元素多過原先配置空間的大小時，**vector** 需移動所有元素到更大的連續空間，**vector** 此時執行效率較差；而 **deque** 可以不佔有連續的儲存空間，當新增元素超過原來配置空間時，只需將新增元素儲存在記憶體其他區塊，不需要搬移舊有的資料，**deque** 會將多個不連續記憶體區塊連結起來，所以新增元素超過原來配置空間時，**deque** 相較 **vector** 有效率。

deque 所提供的重要函式

函式分類	函式	功能	平均效率
迭代器函式	begin()	回傳 deque 的第一個元素	O(1)
	end()	回傳 deque 的最後一個元素的下一個，這個元素不存在，通常用於表示資料已經到達結尾	O(1)
新增或刪除元素函式	push_back(x)	將 x 加到 deque 的最後面。	O(1)
	pop_back()	刪除最後一個元素。	O(1)
	push_front(x)	將 x 加到 deque 的最前面。	O(1)
	pop_front()	刪除第一個元素。	O(1)
	insert(pos,x)	在 deque 的 pos 位置插入元素 x。	O(n)
	erase(pos)	刪除 deque 的 pos 位置的元素。	O(n)
	clear()	刪除 deque 中所有元素	O(n)
讀取元素函式	at(i)或[i]	讀取 deque 第 i 個元素，at(i)在 i 值超出邊界時會發出錯誤訊息，而[i]在 i 值超出邊界時不會發出錯誤訊息。	O(1)
	front()	讀取 deque 第一個元素	O(1)
	back()	讀取 deque 最後一個元素	O(1)
儲存空間函式	empty()	檢查 deque 是否是空的，若是空的，回傳 true，否則回傳 false。	O(1)
	size()	回傳 deque 目前儲存元素的個數	O(1)

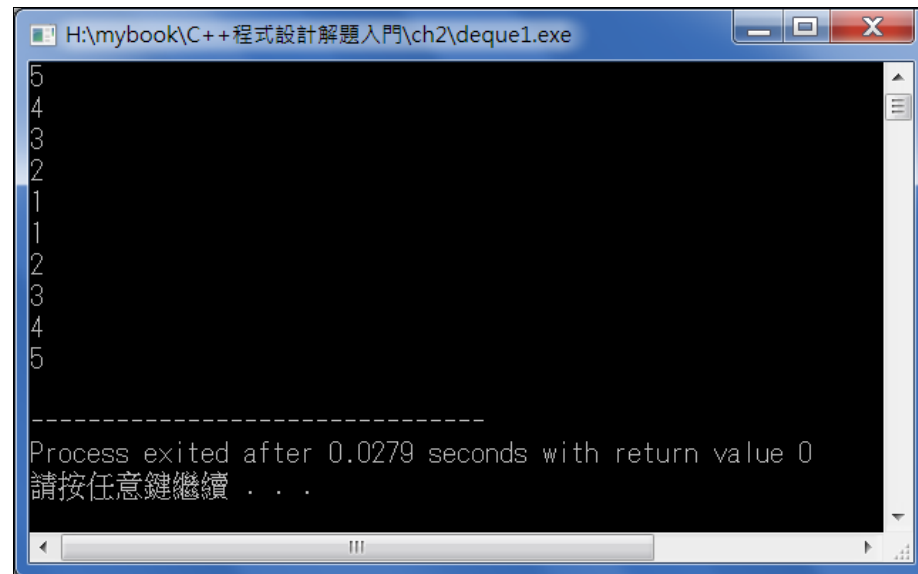
deque 程式範例

A-2-2-在 deque 中新增與讀取元素(chA\A-2-2-deque.cpp)

範例說明

請實作一個程式將數字 1 到 5 由前面依序加入 deque 中，並由前面依序從 deque 中取出顯示到螢幕，之後再將數字 1 到 5 由後面依序加入 deque 中，並由前面依序從 deque 中取出顯示到螢幕。

預期程式執行結果



```
H:\mybook\C++程式設計解題入門\ch2\deque1.exe
5
4
3
2
1
1
2
3
4
5
-----
Process exited after 0.0279 seconds with return value 0
請按任意鍵繼續 . . .
```

範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 dq 為儲存整數(int)的 deque。 第 6 行到第 8 行：使用迴圈依序由前面插入 (dq.push_front(i)) 1、2、3、4 與 5 到 dq。 第 9 行到第 12 行：使用迴圈依序由 dq 的前方取出元素 (dq.front()) 顯示在螢幕上，並將前面的元素刪除 (dq.pop_front())，直到 dq 沒有元素為止 (while(!dq.empty()))。
2	#include <deque>	
3	using namespace std;	
4	int main(){	
5	deque<int> dq;	
6	for(int i=1;i<=5;i++){	第 13 行到第 15 行：使用迴圈依序由後面插入 (dq.push_back(i)) 1、2、3、4 與 5 到 dq。
7	dq.push_front(i);	
8	}	
9	while(!dq.empty()){	第 16 行到第 19 行：使用迴圈依序由 dq 的前方取出元素 (dq.front()) 顯示在螢幕上，並將前
10	cout << dq.front() << endl;	
11	dq.pop_front();	
12	}	

13	for(int i=1;i<=5;i++){	面的元素刪除(<code>dq.pop_front()</code>)，直到 dq 沒有
14	dq.push_back(i);	元素為止(<code>while(!dq.empty())</code>)。
15	}	
16	while(!dq.empty()){	
17	cout << dq.front() << endl;	
18	dq.pop_front();	
19	}	
20	}	

A-2-3 list

雙重鏈結串列實作出 `list`，允許任何位置快速插入與刪除資料，在任何位置插入與刪除資料的成本較 `vector` 與 `deque` 低，但不可以利用索引值直接存取指定的資料，也就是不支援 `at` 函式或 `[]` 運算子。

`list` 所提供的重要函式

函式分類	函式	功能	平均效率
迭代器函式	<code>begin()</code>	回傳 <code>list</code> 的第一個元素	$O(1)$
	<code>end()</code>	回傳 <code>list</code> 的最後一個元素的下一個，這個元素不存在，通常用於表示資料已經到達結尾	$O(1)$
新增或刪除元素函式	<code>push_back(x)</code>	將 <code>x</code> 加到 <code>list</code> 的最後面。	$O(1)$
	<code>pop_back()</code>	刪除最後一個元素。	$O(1)$
	<code>push_front(x)</code>	將 <code>x</code> 加到 <code>list</code> 的最前面。	$O(1)$
	<code>pop_front()</code>	刪除第一個元素。	$O(1)$
	<code>insert(pos,x)</code>	在 <code>list</code> 的 <code>pos</code> 位置插入元素 <code>x</code> ， <code>pos</code> 的值無法任意指定，只能由開始或結束位置，不斷遞增或遞減到指定位置，找到要插入的位置會花較多時間。	$O(1)$
	<code>erase(pos)</code>	刪除 <code>list</code> 的 <code>pos</code> 位置的元素， <code>pos</code> 的值無法任意指定，只能由開始或結束位置，不斷遞增或遞減到指定位置，找到要刪除的位置會花較多時間。	$O(1)$
	<code>clear()</code>	刪除 <code>list</code> 中所有元素	$O(n)$
讀取元素函式	<code>front()</code>	讀取 <code>list</code> 第一個元素	$O(1)$
	<code>back()</code>	讀取 <code>list</code> 最後一個元素	$O(1)$

	<code>remove(x)</code>	將 <code>list</code> 中所有值為 <code>x</code> 的元素刪除	$O(n)$
儲存空間函式	<code>empty()</code>	檢查 <code>list</code> 是否是空的，若是空的，回傳 <code>true</code> ，否則回傳 <code>false</code> 。	$O(1)$
	<code>size()</code>	回傳 <code>list</code> 目前儲存元素的個數	$O(1)$

list 程式範例

A-2-3 在 `list` 中新增與讀取元素(chA\A-2-3-list.cpp)

範例說明

請實作一個程式將數字 1 到 4 由後面依序加入 `list` 中，並使用迭代器(iterator)、遞增運算子(++)與遞減運算子(--)所指定的位置插入數字到 `list`，並使用 `remove` 函式刪除指定數值的所有元素，過程中插入或刪除後，顯示 `list` 所有元素到螢幕。

預期程式執行結果

```

H:\mybook\C++ 程式設計解題入門\ch2\list1.exe
1 2 3 4
1 5 2 3 4
1 4 5 2 3 4
1 5 2 3
-----
Process exited after 0.03305 seconds with return value 0
請按任意鍵繼續...

```

範例程式如下

行號	程式碼	說明
1	<code>#include <iostream></code>	第 5 行：宣告 <code>mlist</code> 為儲存整數(int)的 <code>list</code> 。
2	<code>#include <list></code>	第 6 行：宣告 <code>it</code> 與 <code>its</code> 為指向 <code>list</code> 中元素的迭代器(iterator)。
3	<code>using namespace std;</code>	第 7 行：使用迴圈依序由後面插入
4	<code>int main(){</code>	(<code>mlist.push_back(i)</code>)1、2、3 與 4 到 <code>mlist</code> ，因為迴圈只有一行可以省略一對大括號(<code>{}</code>)。
5	<code>list<int> mlist;</code>	第 8 到 10 行：使用迴圈與迭代器 <code>its</code> ， <code>its</code> 的值
6	<code>list<int>::iterator it, its;</code>	依序由 <code>mlist</code> 開始(<code>mlist.begin()</code>)到結束
7	<code>for (int i=1; i<=4; ++i) mlist.push_back(i); // 1 2 3 4</code>	(<code>mlist.end()</code>)，每次遞增 1，取出每個元素顯示在螢幕上。
8	<code>for (its=mlist.begin(); its!=mlist.end(); ++its){</code>	第 11 行：輸出換行到螢幕。
9	<code>cout << *its << ' ';</code>	
10	<code>}</code>	

11	<code>cout << endl;</code>	第 12 行：設定迭代器 <code>it</code> 指向 <code>mlist</code> 的第一個元素。
12	<code>it = mlist.begin();</code>	第 13 行：設定迭代器 <code>it</code> 遞增 1，指向 <code>mlist</code> 的第二個元素。
13	<code>++it;</code>	第 14 行：在 <code>mlist</code> 的第二個元素位置插入數字 5。
14	<code>mlist.insert(it,5); //1 5 2 3 4</code>	第 15 到 17 行：使用迴圈與迭代器 <code>its</code> ， <code>its</code> 的值依序由 <code>mlist</code> 開始(<code>mlist.begin()</code>)到結束(<code>mlist.end()</code>)，每次遞增 1，取出每個元素顯示在螢幕上。
15	<code>for (its=mlist.begin(); its!=mlist.end(); ++its){</code>	第 18 行：輸出換行到螢幕。
16	<code>cout << *its << ' ';</code>	第 19 行：設定迭代器 <code>it</code> 遞減 1，指向 <code>mlist</code> 的第二個元素。
17	<code>}</code>	第 20 行：在 <code>mlist</code> 的第二個元素位置插入數字 4。
18	<code>cout << endl;</code>	第 21 到 23 行：使用迴圈與迭代器 <code>its</code> ， <code>its</code> 的值依序由 <code>mlist</code> 開始(<code>mlist.begin()</code>)到結束(<code>mlist.end()</code>)，每次遞增 1，取出每個元素顯示在螢幕上。
19	<code>--it;</code>	第 24 行：輸出換行到螢幕。
20	<code>mlist.insert(it,4); //1 4 5 2 3 4</code>	第 25 行：在 <code>mlist</code> 中刪除所有數值為 4 的元素。
21	<code>for (its=mlist.begin(); its!=mlist.end(); ++its){</code>	第 26 到 28 行：使用迴圈與迭代器 <code>its</code> ， <code>its</code> 的值依序由 <code>mlist</code> 開始(<code>mlist.begin()</code>)到結束(<code>mlist.end()</code>)，每次遞增 1，取出每個元素顯示在螢幕上。
22	<code>cout << *its << ' ';</code>	第 29 行：輸出換行到螢幕。
23	<code>}</code>	
24	<code>cout << endl;</code>	
25	<code>mlist.remove(4); //1 5 2 3</code>	
26	<code>for (its=mlist.begin(); its!=mlist.end(); ++its){</code>	
27	<code>cout << *its << ' ';</code>	
28	<code>}</code>	
29	<code>cout << endl;</code>	
30	<code>}</code>	

A-2-4 vector、deque 與 list 的比較

vector 使用連續的空間儲存資料，使用 `push_back` 新增資料時，若遇到空間不夠時，需要花較多時間搬移資料到更大的空間，而 **deque** 與 **list** 因為使用非連續空間儲存資料，不會有這方面問題。**vector** 不支援 `push_front` 與 `pop_front`，而 **deque** 與 **list** 支援 `push_front` 與 `pop_front`。

list 在中間位置插入(`insert`)與刪除(`erase`)元素比 **vector** 與 **deque** 還要有效率，但 **list** 在迭代器部分只能從開始或結束位置，利用遞增(`++`)與遞減(`--`)運算子，連續遞增 1 或遞減 1 直到所要的位置，才能讀取、插入與刪除元素，不支援 `at` 函式或 `[]` 運算子，而 **vector** 與 **deque** 支援 `at` 函式或 `[]` 運算子，可以很快速的指定讀取任意一個元素。

vector、**deque** 與 **list** 的比較，如下表。

資料型別	vector	deque	list
空間	類似陣列，需使用連續空間儲存。	儲存空間不一定要連續	使用雙向鏈結串列
push_back(x)	$O(1)$ ，若超過 vector 預定的大小，需要調整儲存空間時，演算法複雜度為 $O(n)$	$O(1)$	$O(1)$
pop_back()	$O(1)$	$O(1)$	$O(1)$
push_front(x)	不支援	$O(1)$	$O(1)$
pop_front()	不支援	$O(1)$	$O(1)$
insert(pos,x)	$O(n)$	$O(n)$	$O(1)$
erase(pos)	$O(n)$	$O(n)$	$O(1)$
at(i)或[i]	$O(1)$	$O(1)$	不支援

A-3 關聯式容器

關聯式容器為非線性儲存容器，可以快速搜尋資料，可以用於儲存**集合(set)**或**鍵值與儲存值(key 與 value)**配對的資料，分成 **set**、**multiset**、**map** 與 **multimap** 四種關聯式容器。

A-3-1 set(集合)

方便快捷找尋資料是否已經存在，不允許資料重複，在插入資料到 **set** 時，會檢查是否已經有相同資料存在集合內，若已經有相同的資料則不會進行插入該元素，會回傳相同資料在 **set** 所在位置的迭代器(iterator)。插入資料時，**set** 會將資料插入到 **set** 所指定的位置，以二元搜尋樹(binary search tree)實作 **set**，讓 **set** 呈現已排序狀態，因此搜尋、插入與刪除資料時有較好的效率。

set 所提供的重要函式

函式分類	函式	功能	平均效率
迭代器函式	begin()	回傳 set 的第一個元素	O(1)
	end()	回傳 set 的最後一個元素的下一個，這個元素不存在，通常用於表示資料已經到達結尾	O(1)
新增或刪除元素函式	insert(x)	插入元素 x 到 set ，會檢查是否 x 已經在集合內，若 x 已經存在則不會進行插入，會回傳元素 x 在 set 所在位置的迭代器(iterator)	O(log(n))
	erase(it)	刪除 set 的迭代器 it 位置的元素，迭代器 it 的值無法任意指定，只能由開始或結束位置，不斷遞增或遞減到指定位置。	O(1)
	erase(x)	刪除 set 中數值為 x 的元素，回傳刪除幾個元素，回傳值為 0 或 1。	O(log(n))
	clear()	刪除 set 中所有元素	O(n)
操作函式	find(x)	在 set 中找出數值為 x 的元素，會檢查是否 x 已經在集合內，若 x 已經存在則回傳 x 所在的迭代器，否則會回傳 set 的函式 end 所在位置的迭代器。	O(log(n))
	count(x)	計算 set 中數值為 x 的個數，若 x 不在 set 中，則回傳 0，否則回傳 1。	O(log(n))

	<code>lower_bound(x)</code>	set 預設使用遞增排序，若 set 是遞增排序，則是回傳 set 中第一個不小於 x 元素的迭代器；若 set 是遞減排序，則是回傳 set 中第一個不大於 x 元素的迭代器。	$O(\log(n))$
	<code>upper_bound(x)</code>	set 預設使用遞增排序，若 set 是遞增排序，則是回傳 set 中第一個大於 x 元素的迭代器；若 set 是遞減排序，則是回傳 set 中第一個小於 x 元素的迭代器。	$O(\log(n))$
儲存空間函式	<code>empty()</code>	檢查 set 是否是空的，若是空的，回傳 true，否則回傳 false。	$O(1)$
	<code>size()</code>	回傳 set 目前儲存元素的個數	$O(1)$

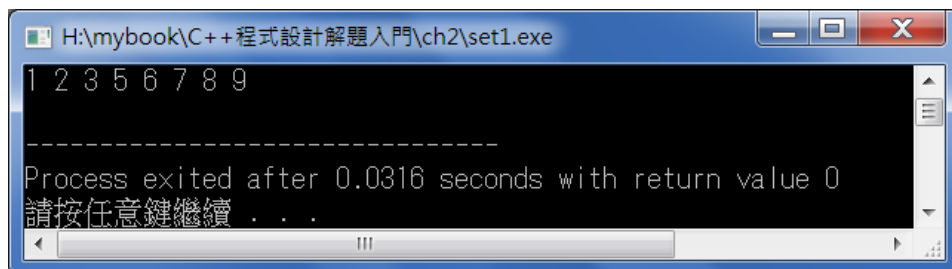
set 程式範例

A-3-1A 在 set 中新增與讀取元素(chA\A-3-1A-set.cpp)

範例說明

請實作一個程式將數字 1 到 9 由依序加入 set 中，使用 find 找出數字 4 所在位置的迭代器(iterator)，並使用 erase 刪除數字 4，並使用迴圈、迭代器(iterator)與遞增運算子(++)，顯示 set 中所有元素到螢幕。

預期程式執行結果



範例程式如下

行號	程式碼	說明
1	<code>#include <iostream></code>	第 5 行：宣告 mset 為儲存整數(int)的 set。
2	<code>#include <set></code>	第 6 行：宣告 itl 與 it 為指向 set 中元素的迭代器(iterator)。
3	<code>using namespace std;</code>	第 7 行：使用迴圈依序由插入
4	<code>int main (){</code>	(<code>mset.insert(i)</code>)1、2、3、...到 9 到 mset，
5	<code>set<int> mset;</code>	因為迴圈只有一行可以省略一對大括號(<code>{}</code>)。
6	<code>set<int>::iterator itl,it;</code>	

7	for (int i=1; i<10; i++) mset.insert(i);	第 8 行：從 mset 找出數字 4，傳回數字 4 在 mset 所在位置的迭代器到 it1。
8	it1=mset.find(4);	第 9 行：從 mset 刪除迭代器 it1 所指的元素。
9	mset.erase(it1);	第 10 行到第 12 行：使用迴圈與迭代器 it，it 的值依序由 mset 開始(mset.begin())到結束(mset.end())，每次遞增 1，取出每個元素顯示在螢幕上。
10	for (it=mset.begin(); it!=mset.end();it++){	第 13 行：輸出換行到螢幕。
11	cout << *it << ' ';	
12	}	
13	cout << endl;	
14	}	

set 預設是遞增排序，若要改成遞減排序，需要在宣告時就告知是遞減排序，使用函式物件(function object)的「**greater<資料型別>**」，宣告成「**set<資料型別,greater<資料型別>>**」，就會變成由大到小排序，注意最後兩個大於「>」中間要插入一個空白鍵，與輸入時所使用的「>>」才能夠區分出來。

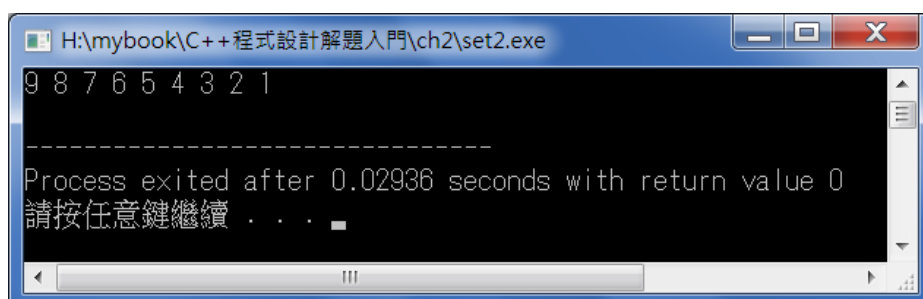
排序方式	宣告語法
遞增排序	set<資料型別>
	set<資料型別,less<資料型別> >
遞減排序	set<資料型別,greater<資料型別> >

A-3-1B 將 set 改成遞減排序(chA\A-3-1B-set.cpp)

範例說明

請實作一個程式將數字 1 到 9 由依序加入 **set** 中，使用遞減排序，並使用迴圈、迭代器(iterator)與遞增運算子(++)，顯示 **set** 中所有元素到螢幕。

預期程式執行結果



範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 mset 為儲存整數(int)的 set，使用遞減排序(<code>greater<int></code>)。
2	#include <set>	
3	using namespace std;	第 6 行：宣告 it 為指向 set 中元素的迭代器(iterator)。
4	int main (){	
5	set<int,greater<int> > mset;	第 7 行：使用迴圈依序由插入(<code>mset.insert(i)</code>)1、2、3、...到 9 到 mset，因為迴圈只有一行可以省略一對大括號(<code>{}</code>)。
6	set<int>::iterator it;	
7	for (int i=1; i<10; i++) mset.insert(i);	第 8 行到第 10 行：使用迴圈與迭代器 it，it 的值依序由 mset 開始(<code>mset.begin()</code>)到結束(<code>mset.end()</code>)，每次遞增 1，取出每個元素顯示在螢幕上。
8	for (it=mset.begin(); it!=mset.end();it++){	
9	cout << *it << ' ';	第 11 行：輸出換行到螢幕。
10	}	
11	cout << endl;	
12	}	

A-3-2 multiset

multiset 與 **set** 的差異在於儲存的資料是否可以重複，**multiset** 允許加入重複的資料，而 **set** 不允許加入重複的資料。插入資料時，**multiset** 會將資料插入到 **multiset** 所指定的位置，以二元搜尋樹(binary search tree)實作 **multiset**，讓 **multiset** 呈現已排序狀態，因此搜尋、插入與刪除資料有較好的效率。

multiset 所提供的重要函式

函式分類	函式	功能	平均效率
迭代器函式	begin()	回傳 multiset 的第一個元素	O(1)
	end()	回傳 multiset 的最後一個元素的下一個，這個元素不存在，通常用於表示資料已經到達結尾	O(1)
新增或刪除元素函式	insert(x)	插入元素 x 到 multiset，會回傳 x 在 multiset 所在位置的迭代器(iterator)	O(log(n))
	erase(it)	刪除 multiset 的迭代器 it 位置的元素，迭代器 it 的值無法任意指定，只能由開始或結束位置，不斷遞增或遞減到指定位置。	O(log(n))
	erase(x)	刪除 multiset 中數值為 x 的元素，回傳刪除幾個元素。	O(log(n))
	clear()	刪除 multiset 中所有元素	O(n)

操作函式	find(x)	在 multiset 中找出數值為 x 的元素，會檢查是否 x 已經在集合內，若 x 已經存在則回傳 x 所在的迭代器，否則會回傳 multiset 的函式 end 所在位置的迭代器。	O(log(n))
	count(x)	計算 multiset 中數值為 x 的個數，若 x 不在 multiset 中，則回傳 0，否則回傳 multiset 中數值 x 的個數。	O(log(n))
	lower_bound(x)	multiset 預設使用遞增排序，若 multiset 是遞增排序，則是回傳 multiset 中第一個不小於 x 元素的迭代器；若 multiset 是遞減排序，則是回傳 multiset 中第一個不大於 x 元素的迭代器。	O(log(n))
	upper_bound(x)	multiset 預設使用遞增排序，若 multiset 是遞增排序，則是回傳 multiset 中第一個大於 x 元素的迭代器；若 multiset 是遞減排序，則是回傳 multiset 中第一個小於 x 元素的迭代器。	O(log(n))
儲存空間函式	empty()	檢查 multiset 是否是空的，若是空的，回傳 true，否則回傳 false。	O(1)
	size()	回傳 multiset 目前儲存元素的個數	O(1)

multiset 程式範例

A-3-2 在 multiset 中新增與讀取元素(chA\A-3-2-multiset.cpp)

範例說明

請實作一個程式將數字由 1 到 5 依序加入 multiset 中，顯示 multiset 中所有元素到螢幕。再一次將數字 1 到 5 依序加入 multiset 中，顯示 multiset 中所有元素到螢幕。使用 erase 刪除 multiset 中所有數字 4 的元素，並使用迴圈、迭代器(iterator)與遞增運算子(++)，顯示 set 中所有元素到螢幕。

預期程式執行結果

```

H:\mybook\C++程式設計解題入門\ch2\multiset1.exe
1 2 3 4 5
1 1 2 2 3 3 4 4 5 5
1 1 2 2 3 3 5 5
-----
Process exited after 0.02749 seconds with return value 0
請按任意鍵繼續 . . .

```

範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 mmset 為儲存整數(int)的 multiset。
2	#include <set>	第 6 行：宣告 it 為指向 set 中元素的迭代器(iterator)。
3	using namespace std;	第 7 行：使用迴圈依序由插入
4	int main (){	(mmset.insert(i))1、2、3、4 與 5 到 mmset，因為迴圈只有一行可以省略一對大括號({})。
5	multiset<int> mmset;	第 8 行到第 10 行：使用迴圈與迭代器 it，it 的值依序由 mmset 開始
6	multiset<int>::iterator it;	(mmset.begin())到結束
7	for (int i=1; i<6; i++) mmset.insert(i);	(mmset.end())，每次遞增 1，取出每個元素顯示在螢幕上。
8	for (it=mmset.begin(); it!=mmset.end();it++){	第 11 行：輸出換行到螢幕。
9	cout << *it << ' ';	第 12 行：使用迴圈依序由插入
10	}	(mmset.insert(i))1、2、3、4 與 5 到 mmset，因為迴圈只有一行可以省略一對大括號({})。
11	cout << endl;	第 13 行到第 15 行：使用迴圈與迭代器 it，it 的值依序由 mmset 開始
12	for (int i=1; i<6; i++) mmset.insert(i);	(mmset.begin())到結束
13	for (it=mmset.begin(); it!=mmset.end();it++){	(mmset.end())，每次遞增 1，取出每個元素顯示在螢幕上。
14	cout << *it << ' ';	第 16 行：輸出換行到螢幕。
15	}	第 17 行：從 mmset 刪除所有數字 4 的元素。
16	cout << endl;	第 18 行到第 20 行：使用迴圈與迭代器 it，it 的值依序由 mmset 開始
17	mmset.erase(4);	(mmset.begin())到結束
18	for (it=mmset.begin(); it!=mmset.end();it++){	(mmset.end())，每次遞增 1，取出每個元素顯示在螢幕上。
19	cout << *it << ' ';	第 21 行：輸出換行到螢幕。
20	}	
21	cout << endl;	
22	}	

A-3-3 map

map 為一對一的映射，映射方式以鍵值(Key)與資料(Value)進行一對一的對應，以鍵值去搜尋所代表的資料，不允許鍵值重複，在插入鍵值與資料到 map 時，會檢查是否已經有相同鍵值存在集合內，若鍵值相同則不會進行插入該元素，會回傳相同鍵值在 map 所在位置的迭代器(iterator)。插入鍵值與資料時，map 會將資料插入到 map 所指定的位置，讓 map 的鍵值(Key)呈現已排序狀態，因此搜尋、插入與刪除資料有較好的效率。map 是以資料結構的二元搜尋樹(binary search tree)進行實作。

map 所提供的重要函式

函式分類	函式	功能	平均效率
迭代器函式	begin()	回傳 map 的第一個元素	O(1)
	end()	回傳 map 的最後一個元素的下一個，這個元素不存在，通常用於表示資料已經到達結尾	O(1)
新增或刪除元素函式	insert(pair<key,value>)	插入元素 pair<key,value>到 map，會檢查是否 key 已經在集合內，若 key 已經存在則不會進行插入，會回傳 key 在 map 所在位置的迭代器(iterator)	O(log(n))
	map[key]	若在 map 中有此 key 值，回傳所對應的 value；若在 map 中沒有此 key 值，就會新增此 key 到 map，沒有對應的 value，就會以預設建構子產生 value。	O(log(n))
	map[key]=value	若在 map 中有此 key 值，以 value 取代原本 key 所對應的資料；若在 map 中沒有此 key 值，就會新增此 key 到 map，此 key 值對應到 value。	O(log(n))
	erase(it)	刪除 map 的迭代器 it 位置的元素，迭代器 it 的值無法任意指定，只能由開始或結束位置，不斷遞增或遞減到指定位置。	O(1)
	erase(key)	刪除 map 中鍵值為 key 的元素，回傳刪除幾個元素，回傳值為 0 或 1。	O(log(n))
	clear()	刪除 map 中所有元素	O(n)
操作函式	find(key)	在 map 中找出數值為 key 的元素，會檢查是否 key 已經在 map 內，若 key 已經存在	O(log(n))

		則回傳 key 所在的迭代器，否則會回傳 map 的函式 end 所在位置的迭代器。	
	count(key)	計算 map 中數值為 key 的個數，若 key 不在 map 中，則回傳 0，否則回傳 1。	$O(\log(n))$
	lower_bound(key)	map 預設使用遞增排序，若 map 是遞增排序，則是回傳 map 中第一個鍵值不小於 key 元素的迭代器；若 map 是遞減排序，則是回傳 map 中第一個鍵值不大於 key 元素的迭代器。	$O(\log(n))$
	upper_bound(key)	map 預設使用遞增排序，若 map 是遞增排序，則是回傳 map 中第一個鍵值大於 key 元素的迭代器；若 map 是遞減排序，則是回傳 map 中第一個小於 key 元素的迭代器。	$O(\log(n))$
儲存空間函式	empty()	檢查 map 是否是空的，若是空的，回傳 true ，否則回傳 false 。	$O(1)$
	size()	回傳 map 目前儲存元素的個數	$O(1)$

map 程式範例

A-3-3A 在 map 中新增與讀取元素(ch\A-3-3A-map. cpp)

範例說明

請實作一個程式將(a,97)、(b,98)、(c,99)、(d,100)依序加入 **map** 中，使用 **find** 找出字元 **c** 所在位置的迭代器(iterator)，並使用 **erase** 刪除字元 **c** 與對應的數值 99，並使用迴圈、迭代器(iterator)與遞增運算子(++)，顯示 **map** 中所有元素到螢幕。

預期程式執行結果

```

H:\mybook\C++ 程式設計解題入門\ch2\map1.exe
a => 97
b => 98
c => 99
d => 100
a => 97
b => 98
d => 100

-----
Process exited after 0.02733 seconds with return value 0
請按任意鍵繼續 . . .

```

範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 mmap 為儲存字元(char)與整數(int)配對的 map。
2	#include <map>	第 6 行：宣告 it 為指向 map 中元素的迭代器(iterator)。
3	using namespace std;	第 7 行：使用 insert 插入(a, 97)到 mmap。
4	int main (){	第 8 行到第 10 行：：使用運算子[]插入(b, 98)、(c, 99)、(d, 100)到 mmap。
5	map<char, int> mmap;	第 11 行到第 13 行：使用迴圈與迭代器 it，it 的值依序由 mmap 開始(mmap.begin())到結束(mmap.end())，每次遞增 1，取出每個元素顯示在螢幕上。
6	map<char, int>::iterator it;	第 14 行：從 mmap 找出字元 c，傳回字元 c 在 mmap 所在位置的迭代器到 it。
7	mmap.insert(pair<char, int>('a', 97));	第 15 行：從 mmap 刪除迭代器 it 所指的元素。
8	mmap['b']=98;	第 16 行到第 18 行：使用迴圈與迭代器 it，it 的值依序由 mmap 開始(mmap.begin())到結束(mmap.end())，每次遞增 1，取出每個元素顯示在螢幕上。
9	mmap['c']=99;	
10	mmap['d']=100;	
11	for (it=mmap.begin(); it!=mmap.end(); ++it){	
12	cout << it->first << " => " << it->second << endl;	
13	}	
14	it=mmap.find('c');	
15	mmap.erase(it);	
16	for (it=mmap.begin(); it!=mmap.end(); ++it){	
17	cout << it->first << " => " << it->second << endl;	
18	}	
19	}	

map 預設是遞增排序，若要改成遞減排序，需要在宣告時就告知是遞減排序，使用函式物件(function object)的「greater<key>」，宣告成「map<key,value,greater<key>>」，就會變成由大到小排序，注意最後兩個大於「>」中間要插入一個空白鍵，與輸入時所使用的運算子「>>」才能夠區分開來。

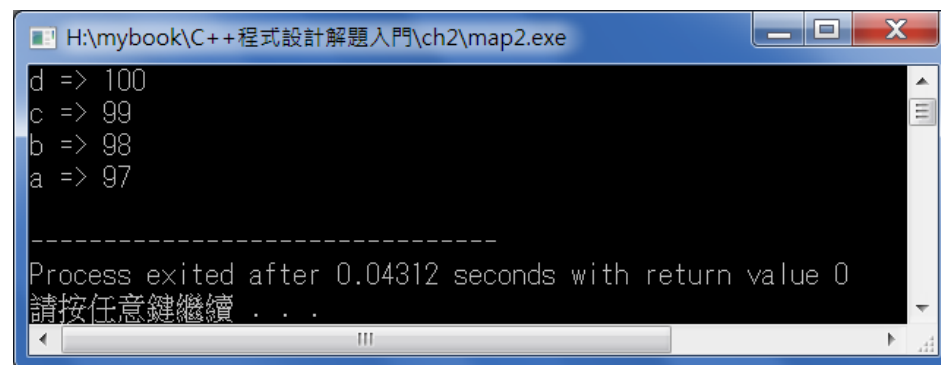
排序方式	宣告語法
遞增排序	map<key,value>
	map<key,value,less<key>>
遞減排序	map<key,value,greater<key>>

A-3-3-B 將 map 改成遞減排序(chA\A-3-3B-map.cpp)

範例說明

請實作一個程式將(a,97)、(b,98)、(c,99)、(d,100)依序加入 map 中，使用遞減排序，並使用迴圈、迭代器(iterator)與遞增運算子(++)，顯示 map 中所有元素到螢幕。

預期程式執行結果



範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 mmap 為儲存字元(char)與整數(int)配對的 map，將字元以遞減排序。
2	#include <map>	
3	using namespace std;	第 6 行：宣告 it 為指向 map 中元素的迭代器(iterator)。
4	int main (){	第 7 行：使用 insert 插入(a, 97)到 mmap。
5	map<char, int, greater<char> > mmap;	
6	map<char, int>::iterator it;	第 8 行到第 10 行：：使用運算子[]插入(b, 98)、(c, 99)、(d, 100)到 mmap。
7	mmap.insert(pair<char, int>('a', 97));	
8	mmap['b']=98;	
9	mmap['c']=99;	第 11 行到第 13 行：使用迴圈與迭代器 it，it 的值依序由 mmap 開始(mmap.begin())到結束(mmap.end())，每次遞增 1，取出每個元素顯示在螢幕上。
10	mmap['d']=100;	
11	for (it=mmap.begin(); it!=mmap.end(); ++it){	
12	cout << it->first << " => " << it->second << endl;	
13	}	
14	}	

A-3-4 multimap

multimap 為一對多的映射，映射方式以鍵值(Key)與資料(Value)進行一對多的對應，以鍵值(Key)去搜尋所代表的資料(Value)，允許鍵值(Key)重複。插入鍵值與資料時，multimap 會將資料插入到 multimap 所指定的位置，讓 multimap 呈現已排序狀態，

因此搜尋、插入與刪除資料有較好的效率。**multimap** 也是以資料結構的二元搜尋樹 (binary search tree) 進行實作。

multimap 所提供的重要函式

函式分類	函式	功能	平均效率
迭代器函式	begin()	回傳 multimap 的第一個元素	O(1)
	end()	回傳 multimap 的最後一個元素的下一個，這個元素不存在，通常用於表示資料已經到達結尾	O(1)
新增或刪除元素函式	insert(pair<key,value>)	插入元素 pair<key,value>到 multimap，會回傳 key 在 multimap 所在位置的迭代器 (iterator)	O(log(n))
	erase(it)	刪除 multimap 的迭代器 it 位置的元素，迭代器 it 的值無法任意指定，只能由開始或結束位置，不斷遞增或遞減到指定位置。	O(1)
	erase(key)	刪除 multimap 中鍵值為 key 的元素，回傳刪除幾個元素。	O(log(n))
	clear()	刪除 multimap 中所有元素	O(n)
操作函式	find(key)	在 multimap 中找出數值為 key 的元素，會檢查是否 key 已經在 multimap 內，若 key 已經存在則回傳 key 所在的迭代器，否則會回傳 multimap 的 end 所在位置的迭代器。	O(log(n))
	count(key)	計算 multimap 中數值為 key 的個數，若 key 不在 multimap 中，則回傳 0，否則回傳 key 的個數。	O(log(n))
	lower_bound(key)	multimap 預設使用遞增排序，若 multimap 是遞增排序，則是回傳 multimap 中第一個鍵值不小於 key 元素的迭代器；若 map 是遞減排序，則是回傳 multimap 中第一個鍵值不大於 key 元素的迭代器。	O(log(n))
	upper_bound(key)	multimap 預設使用遞增排序，若 multimap 是遞增排序，則是回傳 multimap 中第一個鍵值大於 key 元素的迭代器；若 multimap 是遞減排序，則是回傳 multimap 中第一個小於 key 元素的迭代器。	O(log(n))
儲存空間函式	empty()	檢查 multimap 是否是空的，若是空的，回傳 true，否則回傳 false。	O(1)
	size()	回傳 multimap 目前儲存元素的個數	O(1)

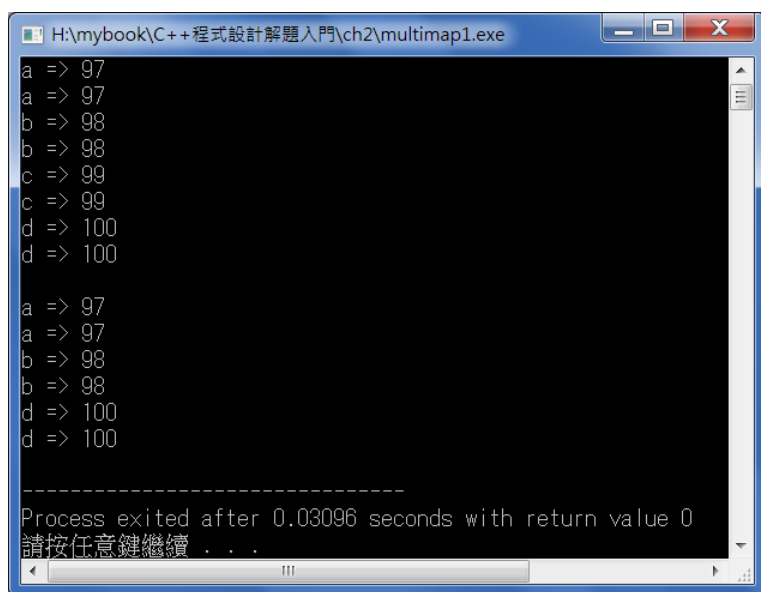
multimap 程式範例

A-3-4 在 multimap 中新增與讀取元素(chA\A-3-4-multimap.cpp)

範例說明

請實作一個程式將(a,97)、(b,98)、(c,99)、(d,100)依序加入 multimap 中，再一次將(a,97)、(b,98)、(c,99)、(d,100)依序加入 multimap 中，並使用 erase 刪除字元 c 與對應的數值 99，並使用迴圈、迭代器(iterator)與遞增運算子(++)，顯示 map 中所有元素到螢幕。

預期程式執行結果



範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 mmap 為儲存字元(char)與整數(int)配對的 multimap。
2	#include <map>	
3	using namespace std;	第 6 行：宣告 it 為指向 multimap 中元素的迭代器(iterator)。
4	int main (){	第 7 行到第 10 行：使用 insert 插入(a, 97)、(b, 98)、(c, 99)、(d, 100)到 mmap。
5	multimap<char, int> mmap;	
6	multimap<char, int>::iterator it;	第 11 行到第 14 行：使用 insert 插入(a, 97)、(b, 98)、(c, 99)、(d, 100)到 mmap。
7	mmap.insert(pair<char, int>('a', 97));	
8	mmap.insert(pair<char, int>('b', 98));	第 15 行到第 17 行：使用迴圈與迭代器 it，it 的值依序由 mmap 開始(mmap.begin())到結束
9	mmap.insert(pair<char, int>('c', 99));	
10	mmap.insert(pair<char, int>('d', 100));	
11	mmap.insert(pair<char, int>('a', 97));	

12	<code>mmap.insert(pair<char,int>('b',98));</code>	(<code>mmap.end()</code>), 每次遞增 1, 取出每個元素顯示在
13	<code>mmap.insert(pair<char,int>('c',99));</code>	螢幕上。
14	<code>mmap.insert(pair<char,int>('d',100));</code>	第 18 行: 輸出換行。
15	<code>for (it=mmap.begin(); it!=mmap.end(); ++it){</code>	第 19 行: 從 mmap 刪除字元 c 的所有元素。
16	<code> cout << it->first << " => " << it->second << endl;</code>	第 20 行到第 22 行: 使用迴圈與迭代器 it, it 的
17	<code>}</code>	值依序由 mmap 開始(<code>mmap.begin()</code>)到結束
18	<code>cout << endl;</code>	(<code>mmap.end()</code>), 每次遞增 1, 取出每個元素顯示在
19	<code>mmap.erase('c');</code>	螢幕上。
20	<code>for (it=mmap.begin(); it!=mmap.end(); ++it){</code>	
21	<code> cout << it->first << " => " << it->second << endl;</code>	
22	<code>}</code>	
23	<code>}</code>	

A-3-5 set、multiset、map 與 multimap 的比較

資料型別	特性
set	能儲存 key 值, 且 key 值不可以重複。
multiset	能儲存 key 值, 且 key 值可以重複。
map	能儲存 key 值與 value 值, 且 key 值不可以重複, 支援以[]運算子讀取與新增元素。
multimap	能儲存 key 值與 value 值, 且 key 值可以重複。

A-4 配接器(adapter)

配接器為序列式容器的變形，限制插入與取出資料是在序列式容器的第一個元素或最後一個元素，分成 **stack**、**queue** 與 **priority_queue** 三種。

A-4-1 stack

stack(堆疊)是後進先出(LIFO)的資料結構，資料的新增與取出都在同一個端點，**stack** 預設使用 **deque** 轉換成 **stack**，也可以指定 **vector** 與 **list** 進行轉換成 **stack**。

組成 stack 的資料結構	宣告語法	說明
deque	<code>stack<int></code>	int 可以取代為任何資料型別，也可以使用自訂的資料型別。
vector	<code>stack<int,vector<int> ></code>	
list	<code>stack<int,list<int> ></code>	

stack 所提供的重要函式

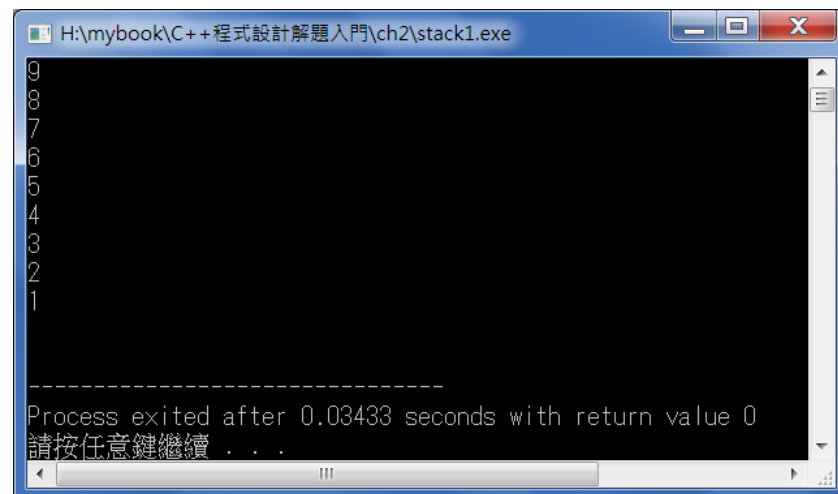
函式分類	函式	功能	平均效率
新增或刪除元素函式	<code>push(x)</code>	插入元素 x 到 stack 的最上面。	$O(1)$
	<code>pop()</code>	從 stack 最上面取出元素，並從 stack 刪除此元素。	$O(1)$
讀取函式	<code>top()</code>	讀取 stack 最上面元素，但不刪除此元素。	$O(1)$
儲存空間函式	<code>empty()</code>	檢查 stack 是否是空的，若是空的，回傳 true ，否則回傳 false 。	$O(1)$
	<code>size()</code>	回傳 stack 目前儲存元素的個數	$O(1)$

A-4-1 在 stack 中新增與讀取元素(chA\A-4-1-stack.cpp)

範例說明

請實作一個程式，將 1、2、3、...與 9 依序加入 **stack** 中，使用迴圈依序取出 **stack** 中所有元素到螢幕。

預期程式執行結果



範例程式如下

行號	程式碼	說明
1	#include <iostream>	<p>第 5 行：宣告 mstack 為儲存整數(int)的 stack。</p> <p>第 6 行：使用迴圈依序由插入 (mstack.push(i))1、2、3、...到 9 到 mstack，因為迴圈只有一行可以省略一對大括號({})。</p> <p>第 7 行到第 10 行：使用 while 迴圈，當 mstack 不是空的，就顯示 mstack 的最上面的元素(mstack.top())到螢幕，將 mstack 的最上面的元素刪除(mstack.pop())，while 迴圈結束後，就會顯示每個元素在螢幕上。</p> <p>第 11 行：輸出換行。</p>
2	#include <stack>	
3	using namespace std;	
4	int main (){	
5	stack<int> mstack;	
6	for (int i=1; i<10; i++) mstack.push(i);	
7	while (!mstack.empty()){	
8	cout << mstack.top() << endl;	
9	mstack.pop();	
10	}	
11	cout << endl;	
12	}	

A-4-2 queue

queue(佇列)是先進先出(FIFO)的資料結構，在一個端點新增資料，在另一個端點取出資料，像排隊買東西一樣，先來的客人先服務。queue 預設使用 deque 轉換成 queue，也可以指定 list 進行轉換成 queue，但無法使用 vector 轉換成 queue。

組成 queue 的資料結構	宣告語法	說明
deque	queue<int>	int 可以取代為任何資料型別，也可以使用自訂的資料型別。
list	queue<int,list<int> >	

queue 所提供的重要函式

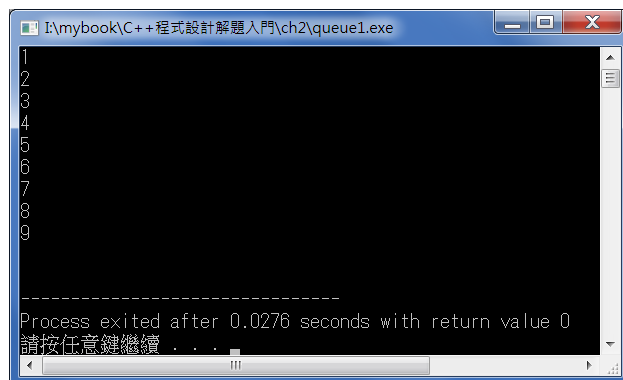
函式分類	函式	功能	平均效率
新增或刪除元素 函式	push(x)	插入元素 x 到 queue 的最後面。	O(1)
	pop()	從 queue 最前面取出元素，並從 queue 刪除此元素。	O(1)
讀取函式	front()	讀取 queue 最前面元素，但不刪除此元素。	O(1)
	back()	讀取 queue 最後面元素，但不刪除此元素。	O(1)
儲存空間函式	empty()	檢查 queue 是否是空的，若是空的，回傳 true，否則回傳 false。	O(1)
	size()	回傳 queue 目前儲存元素的個數	O(1)

A-4-2 在 queue 中新增與讀取元素(chA\A-4-2-queue.cpp)

範例說明

請實作一個程式，將 1、2、3、...與 9 依序加入 queue 中，使用迴圈依序取出 queue 中所有元素到螢幕。

預期程式執行結果



```
I:\mybook\C++程式設計解題入門\ch2\queue1.exe
1
2
3
4
5
6
7
8
9
-----
Process exited after 0.0276 seconds with return value 0
請按任意鍵繼續 . . .
```

範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 mqueue 為儲存整數(int)的
2	#include <queue>	queue。
3	using namespace std;	第 6 行：使用迴圈依序由插入
4	int main (){	(mqueue.push(i))1、2、3、...到 9 到 mqueue，
5	queue<int> mqueue;	因為迴圈只有一行可以省略一對大括號({})。
6	for (int i=1; i<10; i++) mqueue.push(i);	第 7 行到第 10 行：使用 while 迴圈，當 mqueue
7	while (!mqueue.empty()){	不是空的，就顯示 mqueue 的最前面的元素
8	cout << mqueue.front() << endl;	(mqueue.front())到螢幕，將 mqueue 的最前面
9	mqueue.pop();	的元素刪除(mqueue.pop())，while 迴圈結束
10	}	後，就會顯示每個元素在螢幕上。
11	cout << endl;	第 11 行：輸出換行。
12	}	

A-4-3 priority_queue

priority_queue 預設使用 vector 轉換成 priority_queue，也可以使用 deque 轉換成 priority_queue，priority_queue 將 vector 與 deque 轉換成 heap 結構，將最高優先權(最大或最小)的元素優先取出佇列，但不能使用 list 轉換成 priority_queue。

組成 priority_queue 的資料結構	宣告語法	說明
vector	priority_queue<int>	int 可以取代為任何資料型別，也可以自訂資料型別。
deque	priority_queue<int, deque<int> >	

priority_queue 預設是最大的優先取出，若要改成最小的優先取出，需要在宣告時就告知是最小的優先取出，使用函式物件(function object)的「greater<int>」，宣告成「priority_queue<int,vector<int>,greater<int> >」，就會變成由小到大排序，注意最後兩個大於「>」中間要插入一個空白鍵，與輸入時所使用的運算子「>>」才能夠區分開來，所使用資料型別 int，可以宣告為其他資料型別，但資料型別前後要一致。

排序方式	宣告語法	說明
最大的優先取出	priority_queue<int>	int 可以取代為任何資料型別，也可以使用自訂的資料型別。
	priority_queue<int,vector<int>,less<int> >	
最小的優先取出	priority_queue<int,vector<int>,greater<int> >	

priority_queue 所提供的重要函式

函式分類	函式	功能	平均效率
新增或刪除元素函式	push(x)	插入元素 x 到 priority_queue。	O(log(n))
	pop()	從 priority_queue 最上面取出元素，並從 priority_queue 刪除此元素。	O(log(n))
讀取函式	top()	讀取 priority_queue 最上面元素，但不刪除此元素。	O(1)
儲存空間函式	empty()	檢查 priority_queue 是否是空的，若是空的，回傳 true，否則回傳 false。	O(1)
	size()	回傳 priority_queue 目前儲存元素的個數	O(1)

A-4-3A 在 priority_queue 中新增與讀取元素(chA\A-4-3A-priority_queue. cpp)

範例說明

請實作一個程式，將 4、8、3、5 與 2 依序加入 priority_queue 中，使用迴圈依序取出 priority_queue 中所有元素到螢幕。

預期程式執行結果

範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 5 行：宣告 pq 為儲存整數(int)的 priority_queue。
2	#include <queue>	
3	using namespace std;	第 6 行到第 10 行：使用依序由插入 4、8、3、5 與 2 到 pq。
4	int main(){	
5	priority_queue <int> pq; //預設數字大的最上面	第 11 行到第 14 行：使用 while 迴圈，當 pq 不是空的，就顯示 pq 的最上面的元素(pq.top())到螢
6	pq.push(4);	

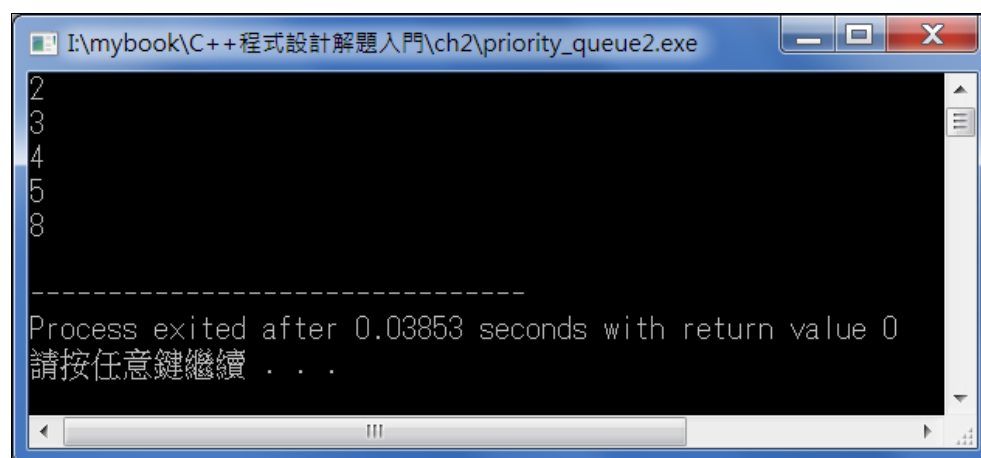
7	<code>pq.push(8);</code>	幕，將 pq 的最上面的元素刪除(<code>pq.pop()</code>)， while 迴圈結束後，就會顯示每個元素在螢幕 上。
8	<code>pq.push(3);</code>	
9	<code>pq.push(5);</code>	
10	<code>pq.push(2);</code>	
11	<code>while(!pq.empty()){</code>	
12	<code> cout << pq.top() << endl;</code>	
13	<code> pq.pop();</code>	
14	<code>}</code>	
15	<code>}</code>	

A-4-3B-將 `priority_queue` 改成遞增排序(chA\A-4-3B-priority_queue.cpp)

範例說明

請實作一個程式，將 4、8、3、5 與 2 依序加入 `priority_queue` 中，此 `priority_queue` 採用遞增排序，使用迴圈依序取出 `priority_queue` 中所有元素到螢幕。

預期程式執行結果



範例程式如下

行號	程式碼	說明
1	<code>#include <iostream></code>	第 5 行：宣告 pq 為儲存整數(int)的 <code>priority_queue</code> ，利用
2	<code>#include <queue></code>	
3	<code>using namespace std;</code>	(<code>greater<int></code>)改成小的數字在最上面。
4	<code>int main(){</code>	第 6 行到第 10 行：使用依序由插入 4、8、3、5 與 2 到 pq。
5	<code> priority_queue <int, vector<int>, greater<int> > pq;</code>	
6	<code> pq.push(4);</code>	

7	<code>pq.push(8);</code>	<p>第 11 行到第 14 行：使用 while 迴圈，當 pq 不是空的，就顯示 pq 的最上面的元素(<code>pq.top()</code>)到螢幕，將 pq 的最上面的元素刪除(<code>pq.pop()</code>)，while 迴圈結束後，就會顯示每個元素在螢幕上。</p>
8	<code>pq.push(3);</code>	
9	<code>pq.push(5);</code>	
10	<code>pq.push(2);</code>	
11	<code>while(!pq.empty()){</code>	
12	<code> cout << pq.top() << endl;</code>	
13	<code> pq.pop();</code>	
14	<code>}</code>	
15	<code>}</code>	

A-5 迭代器(iterator)

迭代器類似指標的功能，可以指定容器中個別的元素，可以與演算法功能一起使用。只有 `vector` 與 `deque` 支援隨機迭代器(random-access iterator)，而 `list`、`set`、`multiset`、`map` 與 `multimap` 支援雙向迭代器(bidirectional iterator)，但配接器 `stack`、`queue` 與 `priority_queue` 不支援迭代器的使用，下面就介紹雙向迭代器與隨機迭代器的差異。

雙向迭代器允許的操作 p 與 q 為迭代器	說明
<code>++p</code>	前置遞增迭代器，先遞增 <code>p</code> 再存取迭代器 <code>p</code> 所在位置
<code>p++</code>	後置遞增迭代器，先存取迭代器 <code>p</code> 所在位置再遞增 <code>p</code>
<code>--p</code>	前置遞減迭代器，先遞減 <code>p</code> 再存取迭代器 <code>p</code> 所在位置
<code>p--</code>	後置遞減迭代器，先存取迭代器 <code>p</code> 所在位置再遞減 <code>p</code>
<code>*p</code>	回傳迭代器所指定元素的資料
<code>p=q</code>	迭代器 <code>p</code> 改成指向迭代器 <code>q</code> 所指的元素
<code>p==q</code>	迭代器 <code>p</code> 是否等於迭代器 <code>q</code>
<code>p!=q</code>	迭代器 <code>p</code> 是否不等於迭代器 <code>q</code>

隨機迭代器功能除了雙向迭代器功能外，還多了以下功能，如下表。

隨機迭代器允許的操作 p 與 q 為迭代器，i 為整數變數	說明
<code>++p</code> 、 <code>p++</code> 、 <code>--p</code> 、 <code>p--</code> 、 <code>*p</code> 、 <code>p=q</code> 、 <code>p==q</code> 與 <code>p!=q</code>	包含雙向迭代器的所有功能
<code>p=p+i</code>	存取迭代器 <code>p</code> 增加 <code>i</code> 個元素的位置，並更新迭代器 <code>p</code>
<code>p=p-i</code>	存取迭代器 <code>p</code> 減少 <code>i</code> 個元素的位置，並更新迭代器 <code>p</code>
<code>p+i</code>	存取迭代器 <code>p</code> 增加 <code>i</code> 個元素的位置，未更新迭代器 <code>p</code>
<code>p-i</code>	存取迭代器 <code>p</code> 減少 <code>i</code> 個元素的位置，未更新迭代器 <code>p</code>
<code>p[i]</code>	存取迭代器 <code>p</code> 增加 <code>i</code> 個元素的資料
<code>p<q</code>	比較 <code>p</code> 是否小於 <code>q</code>
<code>p<=q</code>	比較 <code>p</code> 是否小於等於 <code>q</code>
<code>p>q</code>	比較 <code>p</code> 是否大於 <code>q</code>
<code>p>=q</code>	比較 <code>p</code> 是否大於等於 <code>q</code>

A-6 演算法(algorithm)

演算法(algorithm)提供搜尋、排序與比較等功能，可以將這些功能應用在不同的容器上，讓演算法與資料容器分開，演算法也可以重複使用，增加演算法的使用彈性，減少使用者程式碼的撰寫，且可以使用演算法會配合迭代器存取個別元素。

algorithm 所提供的重要演算法

函式分類	函式	功能	平均效率
找尋演算法	find(first,last,val)	在[first,last)範圍內，範圍包含迭代器 first，但不含迭代器 last，找尋第一個 val，回傳 val 的迭代器，若找不到 val，則回傳迭代器 last。	O(n)
	count(first,last,val)	在[first,last)範圍內，範圍包含迭代器 first，但不含迭代器 last，找尋元素等於 val 的個數，回傳 val 的個數。	O(n)
	search(first1,last1, first2,last2)	在[first1,last1)範圍內，是否依序出現 [first2,last2)範圍內的每個元素，若有出現一模一樣的序列，則回傳[first1,last1)範圍內出現與[first2,last2)一樣序列第一個元素的迭代器，否則回傳迭代器 last1。	O(n*m)，n 為 [first1,last1)範圍內元素個數，m 為 [first2,last2)範圍內元素個數
	binary_search(first, last,val)	已排序的[first,last)範圍內的資料進行二元搜尋，範圍包含迭代器 first，但不含迭代器 last，找尋 val 是否存在，若存在回傳 true，否則回傳 false。	O(log(n))
	lower_bound(first, last,val)	已排序的[first,last)範圍內的資料進行搜尋，範圍包含迭代器 first，但不含迭代器 last，找尋不小於 val 的第一個元素的迭代器，若存在回傳第一個不小於 val 元素的迭代器，否則回傳迭代器 last。	支援隨機迭代器，效率為 O(log(n)) 不支援隨機迭代器，效率為 O(n)
	upper_bound(first, last,val)	已排序的[first,last)範圍內的資料進行搜尋，範圍包含迭代器 first，但不含迭代器 last，找尋大於 val 的第一個元素的迭代器，若存在，則回傳第一個大於 val 元素的迭代器，否則回傳迭代器 last。	支援隨機迭代器，效率為 O(log(n)) 不支援隨機迭代器，效率為 O(n)

排序演算法	<code>sort(first, last)</code>	在 <code>[first,last)</code> 範圍內進行排序，範圍包含迭代器 <code>first</code> ，但不含迭代器 <code>last</code> ，預設使用遞增排序，儲存資料的結構須支援隨機迭代器。	$O(n \cdot \log(n))$
	<code>sort(first, last, cmp)</code>	在 <code>[first,last)</code> 範圍內使用 <code>cmp</code> 比較函式進行排序，範圍包含迭代器 <code>first</code> ，但不含迭代器 <code>last</code> ，經由修改 <code>cmp</code> 比較函式，可以改成遞增排序或遞減排序，儲存資料的結構須支援隨機迭代器。	$O(n \cdot \log(n))$
	<code>stable_sort(first, last)</code>	在 <code>[first,last)</code> 範圍內進行排序，範圍包含迭代器 <code>first</code> ，但不含迭代器 <code>last</code> ，預設使用遞增排序， <code>stable</code> 表示相同數值的元素，在排序過程中前後關係維持不變，儲存資料的結構須支援隨機迭代器。	$O(n \cdot \log(n))$
	<code>stable_sort(first, last, cmp)</code>	在 <code>[first,last)</code> 範圍內使用 <code>cmp</code> 比較函式進行排序，範圍包含迭代器 <code>first</code> ，但不含迭代器 <code>last</code> ，經由修改 <code>cmp</code> 比較函式，可以改成遞增排序或遞減排序， <code>stable</code> 表示相同數值的元素，在排序過程中前後關係維持不變，儲存資料的結構須支援隨機迭代器。	$O(n \cdot \log(n))$
集合演算法	<code>set_intersection(first1, last1, first2, last2, result)</code>	已排序的 <code>[first1,last1)</code> 範圍的資料與已排序的 <code>[first2,last2)</code> 範圍的資料進行交集(兩個集合都存在此元素)，永遠從 <code>[first1,last1)</code> 複製資料到迭代器 <code>result</code> 的儲存容器內，最後已排序的交集結果儲存在迭代器 <code>result</code> 的儲存容器內。	$O(n+m)$ ， n 為 <code>[first1,last1)</code> 範圍內元素個數， m 為 <code>[first2,last2)</code> 範圍內元素個數
	<code>set_union(first1, last1, first2, last2, result)</code>	已排序的 <code>[first1,last1)</code> 範圍的資料與已排序的 <code>[first2,last2)</code> 範圍的資料進行聯集(兩個集合至少一個集合有此元素)，若 <code>[first2,last2)</code> 的元素與 <code>[first1,last1)</code> 的元素相同，則 <code>[first2,last2)</code> 的元素將不會複製到迭代器 <code>result</code> 的儲存容器內，只會複製 <code>[first1,last1)</code> 的元素到迭代器 <code>result</code> 的儲存容器內，最後	$O(n+m)$ ， n 為 <code>[first1,last1)</code> 範圍內元素個數， m 為 <code>[first2,last2)</code> 範圍內元素個數

		已排序的聯集結果儲存在迭代器 result 的儲存容器內。	
	<code>set_difference(first1,last1,first2,last2,result)</code>	已排序的 <code>[first1,last1)</code> 範圍的資料與已排序的 <code>[first2,last2)</code> 範圍的資料進行差集(元素在 <code>[first1,last1)</code> 範圍內找得到，但在 <code>[first2,last2)</code> 範圍內找不到)，永遠從 <code>[first1,last1)</code> 複製資料到迭代器 result 的儲存容器內，最後已排序的差集結果儲存在迭代器 result 的儲存容器內。	$O(n+m)$ ， n 為 <code>[first1,last1)</code> 範圍內元素個數， m 為 <code>[first2,last2)</code> 範圍內元素個數
極值演算法	<code>min(a,b)</code>	回傳 a 與 b 中較小的，若相等回傳 a 。	$O(1)$
	<code>max(a,b)</code>	回傳 a 與 b 中較大的，若相等回傳 a 。	$O(1)$
	<code>min_element(first, last)</code>	在 <code>[first,last)</code> 範圍內，範圍包含迭代器 first ，但不含迭代器 last ，找尋最小元素的迭代器，若有多個最小元素，則回傳第一個找到最小元素的迭代器。	$O(n)$
	<code>max_element(first, last)</code>	在 <code>[first,last)</code> 範圍內，範圍包含迭代器 first ，但不含迭代器 last ，找尋最大元素的迭代器，若有多個最大元素，則回傳第一個找到最大元素的迭代器。	$O(n)$
排列演算法	<code>next_permutation(first, last)</code>	在 <code>[first,last)</code> 範圍內元素為已排序的元素，範圍包含迭代器 first ，但不含迭代器 last ，找尋下一個字母排序更大的排列結果，若找到更大排列結果，則回傳 true ，否則回傳 false 。	$O(n)$

演算法程式範例

A-6-1 stable 排序(chA\A-6-1-stable_sort.cpp)

範例說明

使用 **vector** 初始化為「5.2, 3.4, 3.2, 4.6, 3.3」，使用 **stable_sort** 進行排序，第一次使用預設的浮點數排序，第二次使用自訂的比較函式，將浮點數轉成整數，所以「3.4, 3.2, 3.3」轉成整數都是 3，比較時使用 3 進行排序，看 **stable_sort** 排序後，結果是否為「3.4, 3.2, 3.3」，相同數字排序並不影響原始資料的順序。

預期程式執行結果

```

I:\mybook\C++程式設計解題入門\ch2\stable_sort1.exe
原始資料
5.2 3.4 3.2 4.6 3.3
使用浮點數進行比較與排序
3.2 3.3 3.4 4.6 5.2
將浮點數轉成整數再進行比較與排序
3.4 3.2 3.3 4.6 5.2
-----
Process exited after 0.02452 seconds with return value 0
請按任意鍵繼續 . . .

```

範例程式如下

行號	程式碼	說明
1	<code>#include <iostream></code>	第 5 行到第 7 行：定義 <code>cmpfi</code> 函式為將輸入的浮點數比較，改成整數的比較，會將小數點捨去後進行比較。
2	<code>#include <algorithm></code>	
3	<code>#include <vector></code>	
4	<code>using namespace std;</code>	第 8 行到第 29 行：主程式 <code>main</code> 的程式。
5	<code>bool cmpfi(double i,double j){</code>	第 9 行：宣告 <code>num</code> 為浮點數陣列，初始化為 {5.2, 3.4, 3.2, 4.6, 3.3}。
6	<code>return ((int)i<(int)j);</code>	
7	<code>}</code>	第 10 行：宣告 <code>mvec</code> 為儲存倍精度浮點數 (double) 的 vector。
8	<code>int main () {</code>	第 11 行：宣告 <code>it</code> 為指向 vector 中元素的迭代器(iterator)。
9	<code>double num[] = {5.2, 3.4, 3.2, 4.6, 3.3};</code>	
10	<code>vector<double> mvec;</code>	第 12 行：顯示「原始資料」到螢幕。
11	<code>vector<double>::iterator it;</code>	
12	<code>cout << "原始資料" << endl;</code>	第 13 行：顯示陣列 <code>num</code> 的所有值到螢幕上。
13	<code>for(int i=0;i<5;i++) cout << num[i] << ' ';</code>	
14	<code>cout << endl;</code>	第 14 行：顯示換行。
15	<code>mvec.assign(num,num+5);</code>	第 15 行：將陣列 <code>num</code> 的五個元素複製到 <code>mvec</code> 。
16	<code>stable_sort(mvec.begin(),mvec.end());</code>	
17	<code>cout << "使用浮點數進行比較與排序" << endl;</code>	第 16 行：使用 <code>stable_sort</code> 排序 <code>mvec</code> 。
18	<code>for (it=mvec.begin(); it!=mvec.end(); ++it){</code>	第 17 行：顯示「使用浮點數進行比較與排序」到螢幕。
19	<code>cout << *it << ' ';</code>	
20	<code>}</code>	

21	<code>cout << endl;</code>	第 18 行到第 20 行：使用迴圈與迭代器 <code>it</code> ， <code>it</code> 的值依序由 <code>mvec</code> 開始(<code>mvec.begin()</code>)到結束(<code>mvec.end()</code>)，每次遞增 1，取出 <code>mvec</code> 每個元素顯示在螢幕上。
22	<code>mvec.assign(num, num+5);</code>	第 21 行：輸出換行。
23	<code>stable_sort(mvec.begin(), mvec.end(), cmpfi);</code>	第 22 行：將陣列 <code>num</code> 的五個元素複製到 <code>mvec</code> 。
24	<code>cout<<" 將浮點數轉成整數再進行比較與排序"<<endl;</code>	第 23 行：使用 <code>stable_sort</code> 排序 <code>mvec</code> ，將比較函式改用整數進行比較。
25	<code>for (it=mvec.begin(); it!=mvec.end(); ++it){</code>	第 24 行：顯示「將浮點數轉成整數再進行比較與排序」到螢幕。
26	<code> cout << *it << ' ';</code>	第 25 行到第 27 行：使用迴圈與迭代器 <code>it</code> ， <code>it</code> 的值依序由 <code>mvec</code> 開始(<code>mvec.begin()</code>)到結束(<code>mvec.end()</code>)，每次遞增 1，取出 <code>mvec</code> 每個元素顯示在螢幕上。
27	<code>}</code>	第 28 行：輸出換行。
28	<code>cout << endl;</code>	
29	<code>}</code>	

A-6-2 set 運算(chA\A-6-2-set 運算.cpp)

範例說明

集合 `p` 初始化為{1,3,5,7,9}，集合 `q` 初始化為{2,3,4,5,6}，使用 `set_union` 進行集合 `p` 與 `q` 的聯集運算，使用 `set_intersection` 進行集合 `p` 與 `q` 的交集運算，使用 `set_difference` 進行集合 `p` 與 `q` 的差集運算。

預期程式執行結果

```

I:\mybook\C++ 程式設計解題入門\ch2\set運算.exe
p集合的元素：1 3 5 7 9
q集合的元素：2 3 4 5 6
聯集結果為
1 2 3 4 5 6 7 9
交集結果為
3 5
差集結果為
1 7 9
-----
Process exited after 0.03918 seconds with return value 0
請按任意鍵繼續 . . .

```

範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 6 行：宣告 p 為整數陣列，初始化為{1, 3, 5, 7, 9}。
2	#include <algorithm>	第 7 行：宣告 p 為整數陣列，初始化為{2, 3, 4, 5, 6}。
3	#include <vector>	第 8 行：宣告 u、i 與 d 為儲存整數(int)的 vector，
4	using namespace std;	初始化有 10 個元素，每個元素為 0。
5	int main () {	第 9 行：宣告 it 為指向 vector 中元素的迭代器
6	int p[5] = {1, 3, 5, 7, 9};	(iterator)。
7	int q[5] = {2, 3, 4, 5, 6};	第 10 行：使用 sort 排序陣列 p。
8	vector<int> u(10), i(10), d(10);	第 11 行：使用 sort 排序陣列 q。
9	vector<int>::iterator it;	第 12 行到第 14 行：顯示「p 集合的元素：」，顯示
10	sort(p, p+5);	陣列 p 的所有元素，接著換行。
11	sort(q, q+5);	第 15 行到第 17 行：顯示「q 集合的元素：」，顯示
12	cout << "p 集合的元素：";	陣列 q 的所有元素，接著換行。
13	for(int j=0; j<5; j++) cout << p[j] << " ";	第 18 行：將陣列 p 與陣列 q 進行聯集，將結果儲存到
14	cout << endl;	vector u。
15	cout << "q 集合的元素：";	第 19 行：調整 vector u 的大小，刪除後面多餘的空
16	for(int j=0; j<5; j++) cout << q[j] << " ";	間。
17	cout << endl;	第 20 行：顯示「聯集結果為」與換行到螢幕。
18	it=set_union(p, p+5, q, q+5, u.begin());	第 21 行到第 23 行：使用迴圈與迭代器 it，it 的值依
19	u.resize(it-u.begin());	序由 vector u 開始(u.begin())到結束(u.end())，每
20	cout << "聯集結果為"<<endl;	次遞增 1，取出 vector u 每個元素顯示在螢幕上。
21	for (it=u.begin(); it!=u.end(); ++it){	第 24 行：輸出換行。
22	cout << *it << ' ';	第 25 行：將陣列 p 與陣列 q 進行交集，將結果儲存到
23	}	vector i。
24	cout << endl;	第 26 行：調整 vector i 的大小，刪除後面多餘的空
25	it=set_intersection(p, p+5, q, q+5, i.begin());	間。
26	i.resize(it-i.begin());	第 27 行：顯示「交集結果為」與換行到螢幕。
27	cout << "交集結果為"<<endl;	第 28 行到第 30 行：使用迴圈與迭代器 it，it 的值依
28	for (it=i.begin(); it!=i.end(); ++it){	序由 vector i 開始(i.begin())到結束(i.end())，每
29	cout << *it << ' ';	次遞增 1，取出 vector i 每個元素顯示在螢幕上。
30	}	第 31 行：輸出換行。

31	<code>cout << endl;</code>	第 32 行：將陣列 p 與陣列 q 進行差集運算，將結果儲存到 vector d。
32	<code>it=set_difference(p, p+5, q, q+5, d.begin());</code>	
33	<code>d.resize(it-d.begin());</code>	第 33 行：調整 vector d 的大小，刪除後面多餘的空間。
34	<code>cout << "差集結果為"<<endl;</code>	
35	<code>for (it=d.begin(); it!=d.end(); ++it){</code>	第 34 行：顯示「差集結果為」與換行到螢幕。
36	<code> cout << *it << ' ';</code>	第 35 行到第 37 行：使用迴圈與迭代器 it，it 的值依序由 vector d 開始(d.begin())到結束(d.end())，每次遞增 1，取出 vector d 的每個元素顯示在螢幕上。
37	<code>}</code>	
38	<code>cout << endl;</code>	第 38 行：輸出換行。
39	<code>}</code>	

A-7 應用 STL 解題

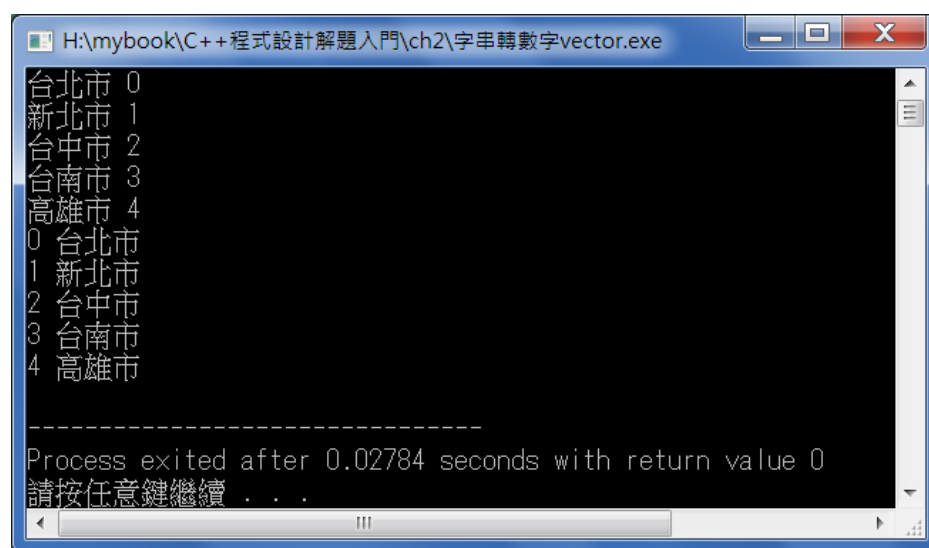
在解題中可以善用 STL 簡化程式碼，不用自行撰寫資料結構與演算法，但須事先了解每個 STL 資料結構與演算法適合應用的地方，以下舉幾個常用的範例供讀者參考。

A-7-1 將城市名稱轉成數字(ch9\9-7-1-字串轉數字 vector.cpp)

範例說明

使用 **vector** 實作一個程式將「台北市」、「新北市」、「台中市」、「台南市」與「高雄市」城市名稱依序轉成 0 到 4 的數字，通常用於將地名轉成數字，建立城市之間的圖形資料結構用，因為城市名稱不適合用於建立圖形資料結構，需將城市名稱轉換成數字，相同的城市名稱要對應相同數字，不同城市名稱對應不同數字。使用二維整數陣列就可以記錄圖形是否有邊相連或儲存邊的權重，如此建立圖形資料結構。使用 **vector** 實作將字串轉成編號，可以將編號再轉換成字串，下一節使用 **map** 將字串轉成編號，有較好的執行效率，只能將字串轉換成編號，但無法將編號再轉換成字串。

預期程式執行結果



範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 6 行：宣告 name 為儲存字串(string)的 vector。 第 7 行：宣告 it 為指向 vector 元素的迭代器(iterator)。 第 8 行到第 15 行：自訂 getCityIndex 函式，從 name 中找出是否曾經出現過 p，結果儲存到
2	#include <algorithm>	
3	#include <string>	
4	#include <vector>	
5	using namespace std;	
6	vector<string> name;	

7	vector<string>::iterator it;	迭代器 it，若迭代器 it 小於 name 的迭代器
8	int getCityIndex(string p){	end，表示 name 中找到字串 p，回傳迭代器 it
9	it=find(name.begin(),name.end(),p); //在 name 中找尋字串 p	減去 name 的迭代器 begin，也就是回傳 p 在
10	if (it < name.end()) return it-name.begin();	name 的編號(第 10 行)。若迭代器 it 等於 name
11	else if (it == name.end()){ //找不到	的迭代器 end，表示 name 中找不到字串 p，則
12	name.push_back(p); //字串 p 加入 name	將字串 p 加到 name 中，回傳 name 的個數減去
13	return name.size()-1;	1，表示字串 p 的編號(第 11 行到第 14 行)。
14	}	第 16 行到第 25 行：定義 main 函式。
15	}	第 17 行：宣告字串 s 有五個元素，分別是「台
16	int main(){	北市」、「新北市」、「台中市」、「台南
17	string s[5]={"台北市","新北市","台中市","台南市","高雄市"};	市」、「高雄市」。
18	for(int i=0;i<5;i++){	第 18 行到第 21 行：使用迴圈依序輸入 s[0]到
19	int index=getCityIndex(s[i]);	s[4]到函式 getCityIndex，回傳城市名稱 s[i]
20	cout << s[i] << " " << index <<endl;	到整數變數 index，顯示城市名稱 s[i]與整數
21	}	變數 index 到螢幕。
22	for(int i=0;i<5;i++){	第 22 行到第 24 行：使用迴圈依序顯示索引值 i
23	cout << i <<" " << name[i] <<endl;	與城市名稱 name[i]到螢幕。
24	}	
25	}	

A-7-2 將城市名稱轉成數字(chA\A-7-2-字串轉數字 map.cpp)

範例說明

使用 **map** 實作一個程式將「台北市」、「新北市」、「台中市」、「台南市」與「高雄市」城市名稱依序轉成 0 到 4 的數字，通常用於將地名轉成數字，建立城市之間的圖形資料結構用，因為城市名稱不適合用於建立圖形資料結構，需將城市名稱轉換成數字，相同的城市名稱要對應相同數字，不同城市名稱對應不同數字。使用二維整數陣列就可以記錄圖形是否有邊相連，建立圖形資料結構。城市名稱儲存到 **map** 後，會自動排序城市名稱，搜尋城市名稱是否存在於 **map** 中會比 **vector** 效率要好，但只能將字串轉換成編號，但無法將編號再轉換成字串。

預期程式執行結果

```

H:\mybook\C++\程式設計解題入門\ch2\字串轉數字map.exe
台北市 0
新北市 1
台中市 2
台南市 3
高雄市 4
台中市 2
台北市 0
台南市 3
高雄市 4
新北市 1
-----
Process exited after 0.02801 seconds with return value 0
請按任意鍵繼續 . . .

```

範例程式如下

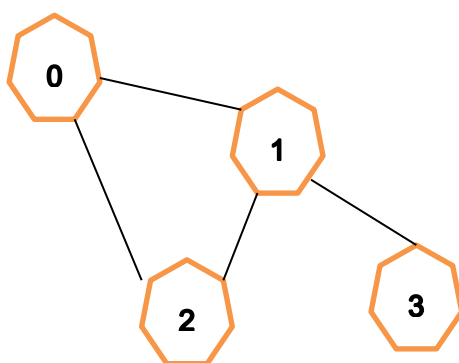
行號	程式碼	說明
1	#include <iostream>	第 6 行：宣告 nmap 為儲存字串(string)與整數(int)配對的 map。
2	#include <algorithm>	
3	#include <string>	第 7 行：宣告 it 為指向 map 中元素的迭代器(iterator)。
4	#include <map>	
5	using namespace std;	第 8 行到第 14 行：自訂 getCityIndex 函式，從 nmap 中找出是否曾經出現過 p，若找尋結果等於 nmap 的迭代器 end，表示 nmap 中找不到字串 p，則變數 s 令為 nmap 的元素個數，表示字串的編號，將字串 p 加到 nmap 中，且編號為 s，回傳 nmap[p] 所對應的值(第 11 行)，表示字串 p 的編號。
6	map<string,int> nmap; //使用 map 將 name 轉成數字	
7	map<string,int>::iterator it;	
8	int getCityIndex(string p){ //將 name 轉成數字，數字從 0 開始	
9	if (nmap.find(p)==nmap.end()){	
10	int s=nmap.size();	
11	nmap[p]=s;	
12	}	
13	return nmap[p];	
14	}	
15	int main(){	第 15 行到第 24 行：定義 main 函式。
16	string s[5]={"台北市","新北市","台中市","台南市","高雄市"};	
17	for(int i=0;i<5;i++){	第 16 行：宣告字串 s 有五個元素，分別是「台北市」、「新北市」、「台中市」、「台南市」、「高雄市」。
18	int index=getCityIndex(s[i]);	
19	cout << s[i] << " " << index << endl;	
20	}	
	}	

21	<code>for(it=nmap.begin();it!=nmap.end();it++){</code>	第 22 行到第 24 行：使用迴圈依序顯示編號與城市名稱到螢幕。
22	<code> cout << it->first << " " << it->second << endl;</code>	
23	<code> }</code>	
24	<code>}</code>	

A-7-3 建立圖形(chA\A-7-3-使用 deque 建圖.cpp)

範例說明

實作一個程式，將以下無向圖儲存到 **deque** 陣列中，未來在圖形結構單元就可以使用此 **deque** 陣列，進行圖形的深度優先走訪或廣度優先走訪。若輸入的點是字串名稱也沒關係，可以使用前兩節 A-7-1 與 A-7-2 轉換成編號，再建立圖形。



我們要利用 **deque** 陣列的每個元素紀錄可以連出去的點，假設 **deque** 陣列名稱為 **F**，以本圖形而言，**F[0]**表示點 0 可以連出去的點就加到 **F[0]**裡，**F[0]**會有 1 與 2，**F[1]**會有 0、2 與 3，**F[2]**會有 0、1，**F[3]**會有 1，將圖形結構轉成 **deque** 陣列，走訪此圖形就相當於走訪對應的 **deque** 陣列。

輸入範例

```

4
0 1
0 2
1 2
1 3

```

預期程式執行結果

範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 4 行：宣告陣列 F 為儲存整數(int)的 deque，陣列每一個元素都是 deque，如此陣列 F 其實是儲存可以連出去點的二維的陣列。
2	#include <deque>	
3	using namespace std;	第 6 行：宣告 x、y 與 m 為整數。
4	deque<int> F[4];	
5	int main(){	第 7 行：輸入圖形邊的個數到變數 m。
6	int x,y,m;	
7	cin >> m;	第 7 行到第 12 行：使用 for 迴圈執行 m 次，每次輸入兩個數字到 x 與 y，表示 x 與 y 有邊相連，因為邊是無向邊，所有兩個方向都要加入，也就是 F[x]需要加入點 y，表示 x 連到 y，F[y]需要加入點 x，表示 y 連到 x。
8	for(int i=0;i<m;i++){	
9	cin >> x >> y;	第 13 行：輸出換行。
10	F[x].push_back(y);	
11	F[y].push_back(x);	第 14 行到第 20 行：使用巢狀迴圈，外層迴圈變數 i，從 0 到 3，每次遞增 1，表示依序取出出發點，顯示出發點(第 15 行)，內層迴圈變數 j，從 0 到 F[i].size()-1，每次遞增 1，取出 F[i]的每一個元素，利用 F[i][j]進行取出元素顯示在螢幕上。每輸出一個出發點後，進行換行(第 19 行)。
12	}	
13	cout << endl;	
14	for(int i=0;i<4;i++){	
15	cout << i << "=>";	
16	for(int j=0;j<F[i].size();j++){	
17	cout << F[i][j] << " ";	
18	}	
19	cout << endl;	
20	}	
21	}	

A-7-4 找出所有單字(chA\A-7-4-使用 set 儲存單字.cpp)

範例說明

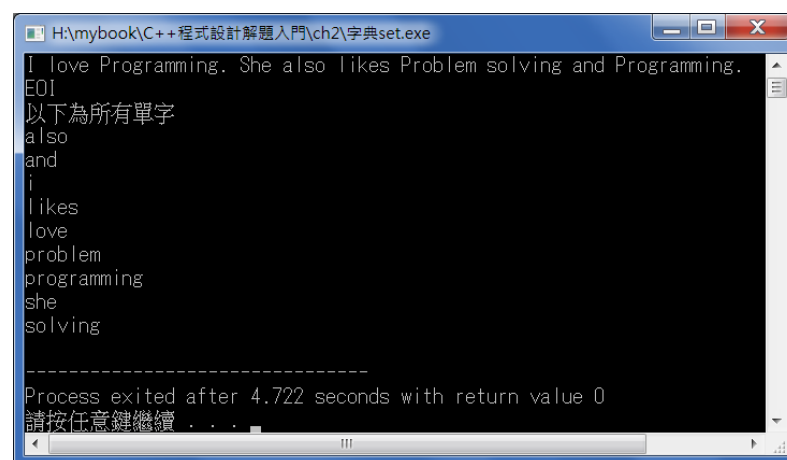
實作一個程式，將文章中所有單字轉成小寫字母儲存到 **set**，按照字母順序列印出所有單字，當輸入 **EOI** 結束輸入，顯示 **set** 中所有單字到螢幕。

輸入範例

I love Programming. She also likes Problem solving and Programming.

EOI

預期程式執行結果



範例程式如下

行號	程式碼	說明
1	#include <iostream>	第 7 行：宣告 dic 為儲存字串(string)的 set。
2	#include <set>	
3	#include <string>	第 8 行：宣告 it 為指向 set 中元素的迭代器 (iterator)。
4	#include <cctype>	
5	using namespace std;	第 9 行：宣告 word 為字串變數。
6	int main(){	第 10 行到 20 行：以 while 迴圈不斷輸入字串到變數 word。當變數 word 等於「EOI」結束迴圈(第 11 行)。使用 for 迴圈讀取變數 word 中每個字元若是英文字母，則一律轉成小寫字母(第 13 行到 14 行)，將字插入到 dic 集合內(第 17 行)。
7	set<string> dic;	
8	set<string>::iterator it;	
9	string word;	
10	while(cin>>word){	
11	if (word == "EOI") break;	第 19 行：輸出「以下為所有單字」。
12	for(int i=0;i<word.length();i++){	
13	if (isalpha(word[i])){	

14	word[i]=tolower(word[i]);	第 20 行到第 22 行：使用迴圈與迭代器 it，it 的值依序由 dic 開始(<code>dic.begin()</code>)到結束(<code>dic.end()</code>)，每次遞增 1，取出 dic 中每個元素顯示在螢幕上。
15	}	
16	}	
17	dic.insert(word);	
18	}	
19	cout << "以下為所有單字" << endl;	
20	for(it=dic.begin();it!=dic.end();it++){	
21	cout << *it<<endl;	
22	}	
23	}	

線上解題系統題目資源

大數問題		
分類	UVa題目	分類
基礎題	UVa 10815 Andy's First Dictionary	set
基礎題	UVa 10391 Compound Words	set
基礎題	UVa 156 Ananagrams	map
進階題	UVa 1592 Database	map
進階題	UVa 814 The Letter Carrier's Rounds	map

註 1：queue、stack 與 priority queue 相關線上解題系統題目資源，參閱第 8 章。

註 2：使用題目編號與名稱為關鍵字進行搜尋，可以找到許多網路資源。