15. STL in C++

C++之 Standard Template Library (STL)為使用樣版功能所製作的標準類別庫，其主要包含了一組容器類別(container classes)，可用於物件之儲存(storing)、管理(managing)、存取(accessing)與操作(manipulating)。

15.1 Container Classes in STL

C++之 STL 包含了以下 10 種容器類別：

1. vector
   可於末端有效率地快速加入或刪除元素，其於使用' [ ] '運算子隨機存取元素時特別有效率。

2. list
   可於任何的位置有效率地快速加入或刪除元素，包含首端與末端，但不支援以' [ ] '運算子隨機存取元素，尋找元素需 scan。

3. stack
   只能由末端作存取，提供後進先出(last-in-first-out)之資料結構。

4. queue
   只能由前端作存取，提供先進先出(first-in-first-out)之資料結構。

5. priority_queue
   元素依其優先權之值(priority value)之次序排序。

6. deque
   名稱由'double-ended queue'與'check'之諧音而來，其結合了 vector 與 list 之優點，可如 list 由首端與末端快速地加入或刪除元素，以及可如 vector 使用' [ ] '運算子快速地隨機存取任何元素，但對於於任意位置之元素增刪運算則與 vector 一樣不如 list 有效率。

   故與 vector 比起來，其多提供了有效率的前端存取；與 list 比起來，其多提供了有效率的隨機存取(使用[ ])。

7. map
   儲存資料為(key, value)之鍵值與對應值之型式，稱為 pair，於使用 key 值查取其對應之 value 值時特別有效率，資料依 key 值排序(< by default)，key 值不能重複。例如以 map 儲存一電話簿，其中人名不重複，則人名為 key，電話號碼為 value，此 map 可宣告為： map<string, int> phonebook;

8. multimap
   更通常化的 map，允許重複的 key 值，亦即允許一對多的映射，例如人名可重複的電話簿(一人可有多支電話)。

9. set
   特殊的 map，其為只有 key 而無對應值的 map 資料結構，可視為值(key)不可重複，並排序儲存之 vector 或 list。

10. multiset
    特殊的 multimap，其為只有 key 而無對應值的 multimap 資料結構，可視為值(key)可重複，並排序儲存之 vector 或 list。

15.2 Common Facilities Offered by All STL Containers

STL 之所有容器類別至少提供以下之介面/功能：

(a) default constructor
(b) copy constructor
(c) assignment operator
(d) ==, !=, <, >, <=, >= 等運算子
(e) empty( ) //returns true if container is empty
(f) size( ) //returns the number of elements in container
(g) max_size( ) //returns the maximum possible size for container
(h) swap( ) //swaps contents with other container

15.3 Iterators

iterator 之於容器類別有如指標之於陣列，其主要功能如下。

1. 支援使用「*」運算子做依址取直(deferencing)之運算，傳回 iterator 目前所指

之元素。

2. 支援「++」運算，將 iterator 指到序列中之下一個元素。
3. 支援「==」運算，當兩個 iterators 相等(==)，則它們指到記憶體中同一個物件。

除了 stacks、queues、與 priority_queues(這三種屬於 adapters)，其他容器類別均提供有 iterators，以用於其儲存物件之存取，某容器類別之 iterator 宣告方式如下：

```
container<type>::iterator    iter;
```

其中，container 為 vector、list 等容器類別名稱
，type 為類別或內建型別名稱
，iterator 為以下之一：

```
iterator                //forward iterator
reverse_iterator        //reverse iterator(++ goes backwards)
const_iterator          //forward iterator, cannot modify elements
const_reverse_iterator  //reverse iterator, cannot modify elements
```

例：

```
vector<string>::iterator    p;              //iterator for vector<string>
list<int>::iterator    iter;                //iterator for list<lnt>
list<string>::reverse_iterator    r;        //reverse iterator for list<string>
map<string, double>::iterator    miter;     //iterator for map<string, double>
```

15.4 Categorization of Container Types

此十種容器類別可分為以下三類。

(1) Sequenced containers:
vector, list, deque

(2) Associative containers:
map, multimap, set, multiset

(3) Adapters:

stack, queue, priority_queue

STL 針對所有 sequenced 與 associative containers 提供並最佳化了以下的運算：

(a) begin( ) //returns iterator to the first element
(b) end( ) //returns iterator to one after last element
(c) rbegin( ) //returns reverse iterator to the last element
(d) rend( ) //returns reverse iterator to one before the first element
(e) insert(iterator, value) //inserts value just before the position specified by iterator
(f) erase(iterator) //erase element at the location specified by iterator

STL(只)針對所有 sequenced containers 提供並最佳化了以下的運算：

(a) front( ) //returns a reference to the first element
(b) back( ) //returns a reference to the last element
(c) push_back(value) //appends value to the container
(d) pop_back( ) //erases the last element of the container

STL(只)針對所有 associative containers 提供並最佳化了以下的運算：

(a) key_comp( ) //returns comparison object
(b) value_comp( ) //returns comparison object
(c) find(key) //look for element based on key
(d) lower_bound( ) //look for first position to insert
// so that the order remains consistent
(e) upper_bound( ) //look for last position to insert
// so that the order remains consistent
(f) count(key) //return number of elements that match key

最後，STL(只)針對所有 adaptors 提供並最佳化了以下的運算：

(a) push(value) //adds element
(b) pop( ) //erases element

15.5 Element Requirements

容器內別內之元素(elements)為插入物件之拷貝，因此，物件須支援對其拷貝之運算，才能被存於容器類別，容器類別使用物件之拷貝建構函式或拷貝指定運算子，產生其所存元素。

當存物件之拷備並不正確時，可使用儲存物件指標的方式，亦即以物件之指標當作容器類別之元素。

15.6 Vectors

vector 容器類別定義於<vector>標題檔內，STL 函式庫並沒有「.h」，以與傳統的函式庫區別，使用 STL 時需加上 using namespace std;之宣告。

以下為使用 vector 之範例：

(1) Summing int vector in C++

程式範例: oop_ex132.cpp

注意要點:

1. Visual C++並沒有 vector.h。
2. STL 提供了新的<iostream>，以對 STL 中的類別，例如 string，提供較好的支援。

(2) List operations on vectors

程式範例: oop_ex133.cpp

注意要點:

1. C++之 vector 亦支援 list operations，如 insert，但效率不如 list。
2. sort( )、find( ) 、find_if( )定義於<algorithm>。

(3) Using an array to initialize a vector

程式範例: oop_ex134.cpp

(4) Vector resizing and its consequences

程式範例: oop_ex135.cpp

15.6.1 Vectors in Java

Java 亦提供了 Vector 之資料結構，其元素均為 Object 之參考，由於 Java 所有類別均繼承自 Object 類別，故基於多型，可用來儲存每一種類別之物件，且同時可儲存不同種類之物件。

程式範例: oop_ex136.java

1. Vector is in java.util.*。
2. 存取使用 Vector 中之物件，需將其由 Object 之參考，cast 回其原本之型態。

15.7 Lists

程式範例: oop_ex137.cpp

注意要點: List 容器類別定義於<vector>標題檔內，其要點如下：

1. No subscripting ([ ]) allowed.

2. sort( )為其函式成員之一，以下程式將 alist 中之元素依預設之「<」運算結果排序(由小而大)。

```
list<string> alist;
alist.sort( );
```

我們亦可以自訂的比較函式取代「<」作為比較的基準，例如自訂的比較函式為 Cmp，則：

```
alist.sort(Cmp);
```

3. splice( )為其函式成員之一，可將某元素自一 list 中取出，再加入另一個 list，例如將 citrus 的第一個元素取出，於 p 所指元素前插入 fruit 中。

```
list<string>::iterator p = find(….);
fruit.splice(p, citrus, citrus.begin());
```

我們亦可將 citrus 全部元素取出，於 p 所指元素前插入 fruit 中：

```
fruit.splice(p, citrus);
```

值得一提的是，splice 指令並非將元素自一 list 拷貝到另一個，而是以調整節點之指標的方式，有效率地達成。

4. merge( )為其函式成員之一，可將兩排序過的 list 結合為一，二者元素的位置彼此穿插，維持結合後元素仍依次序排列。

```
fruit.sort( );
citrus.sort( );
fruit.merge(citrus);
```

如其中的某一個 list 未排序，則 merge( )之結果仍為兩個 list 之集合，但元素不做排序。

與 splice 相同，merge 指令並不做元素之拷貝，而是以調整節點之指標的方式，有效率地達成。

5. List 支援以下之前端運算(front operation)。

```
T& front( );    //returns a reference to the first element of the list

void push_front(const T& item);
void pop_front( );
```

6. List 支援以下之末端運算(back operation)。

```
T& back( );    //returns a reference to the last element of the list
```

```
void push_back(const T& item);
void pop_back( );
```

7. List 支援之其他運算。

```
remove( )        //removes a specific element
remove_if( )     //removes elements satisfying a criterion
unique( )        //removes duplicates
reverse( )       //reverses the order of elements
```

15.7 Maps

一 Map 為「pair」之集合，pair 為鍵值(key)與其對應值(value)所構成之結構體，如下：

```
template<class T1, class T2>
struct pair
{
    T1 first;
    T2 second;
    pair() : first( T1()), second( T2()){ }
    pair(const T1& x, const T2& y) : first( x), second( y){ }
    …..
}
```

Map 於使用鍵值(key)做為索引查詢其對應值時很有效率，例如：

```
map<string, int> phonebook;
//…….
string n;
cin>>n;
cout<<"his or her TEL is "<<phonebook[n]<<endl;
```

程式範例: **oop_ex138.cpp, oop_ex139.cpp**