標準樣板函式庫
(Standard Template Library)

講師：洪安

大綱

- 標準樣板函式庫 (STL)
  - 抽象指標 (Iterator)
  - 容器 (Container)
  - 演算法 (Algorithm)

標準樣板函式庫 (STL)

- STL (Standard Template Library)
- 提供多種類別樣板以供使用
- 由以下三大部分所組成
  - Iterator (抽象指標)
  - Container (容器):
    - vector, list, stack, queue, map,…
  - Algorithm (演算法):
    - find, sort, count,…

大綱

- 標準樣板函式庫 (STL)
  - 抽象指標 (Iterator)
  - 容器 (Container)
  - 演算法 (Algorithm)

抽象指標(Iterator)

- Iterator 是一種可以讓使用者逐一存取 container 中元素的工具
- 所有 STL 的 container C 都有(甚至自己寫的 class)
- 使用者均可利用 C::iterator it; 的方式來宣告 iterator ，以取得容器內元素的位址
- 容器均有提供 begin( ) , end( ) 成員函數，讓使用者取得容器的第一個元素的位址與最後一個元素後一個的位址

抽象指標(Iterator)

- 範例:

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

int main()
{
   int a[7] = { 8, 1, 3, 2, 5, 1, 4};
   vector<int> v(a,a+7);
   vector<int>::iterator it ;
   for ( it = v.begin( ); it != v.end( ) ; it++)
      cout << *it <<" ";
   cout << endl;

   return 0;
}
```

大綱

- 標準樣板函式庫 (STL)
  - 抽象指標 (Iterator)
  - 容器 (Container)
  - 演算法 (Algorithm)

容器(Containers)

- Vector
  - 動態陣列，此陣列可任意增長其大小，並提供隨機存取
- List
  - 雙向linked list，支援循序雙向存取(無法隨機存取)
- Map
  - 支援查表功能之資料結構，資料均以key/value方式存入
- 其他… (stack, queue, set…)

向量 vector

- vector其特性如同陣列一樣，但又比內建的陣列多了許多好用的功能
  - vector可以動態成長，可以將一個vector指定給另一個vector
  - vector知道自己內含元素的個數

常用函式	
[n]	取得第索引值n之資料
size	計算長度
front	取得開頭元素
back	取得結尾元素
pop_back	移除結尾資料
push_back	新增資料於結尾
begin	取得開頭元素之抽象指標
end	取得結尾元素之抽象指標
insert	插入資料
erase	刪除資料

vector::vector

- 範例:

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

int main ()
{
  int i;
  int myints[] = {16,2,77,29};
  vector<int> v1 (myints, myints + 4);
  vector<int> v2;
  v2 = v1;
  for (i=0; i < v2.size(); i++)
    cout << " " << v2[i];
  cout << endl;

  return 0;
}
```

vector::pop_back, push_back

- 範例:

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

int main ()
{
  vector<int> v;
  int sum=0;
  v.push_back (100);
  v.push_back (200);
  v.push_back (300);
  while (!v.empty())
  {
    sum+=v.back();
    v.pop_back();
  }

  cout << "sum = " << sum << endl;
  return 0;
}
```

vector::begin, end

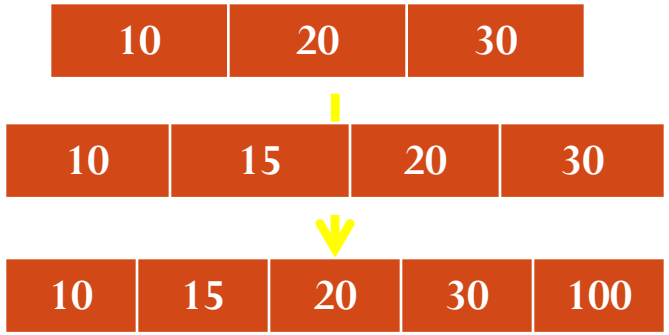
- 範例:

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

int main ()
{
  vector<int> v;
  vector<int>::iterator it;
  for (int i=1; i<=5; i++)
    v.push_back(i);
  for ( it=v.begin() ; it < v.end(); it++ )
   cout << " " << *it;
  cout << endl;

  return 0;
}
```

vector::insert

- 範例:

```
// inserting into a vector
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  int i;
  vector<int> v;
  int a[] = { 10,20,30 };

  v.insert (v.begin(), a, a+3);  //插入一段資料於開頭
  v.insert (v.begin()+1,15);  //插入一筆資料於第二個位子
  v.insert (v.end(),100); //插入一筆資料於最後
  for (i=0; i<v.size(); i++)
    cout << " " << v[i];
  cout << endl;

  return 0;
}
```



begin	begin+1	begin+2	begin+3	begin+4	begin+5	begin+6	begin+7	begin+8	begin+9
1	2	3	4	5	6	7	8	9	10

vector::erase

- 範例:

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

int main ()
{
  int i;
  vector<int> v;

  for (i=1; i<=10; i++)
    v.push_back(i);
  v.erase (v.begin()+5);  //指定一筆資料刪除
  v.erase (v.begin(),v.begin()+3); //刪除一段資料

  for (i=0; i<v.size(); i++)
   cout << " " << v[i];
  cout << endl;

  return 0;
}
```

串列 list

- list其特性主要為實作串列資料結構

常用函式	
size	計算長度
front	取得開頭元素
back	取得結尾元素
begin	取得開頭元素之抽象指標
end	取得結尾元素之抽象指標
pop_back, (pop_front)	移除結尾(開頭)資料
push_back, (push_front)	新增資料於結尾(開頭)
insert	插入資料
erase	刪除資料
remove	指定一個資料內容,並刪除
reverse	資料反置
merge	合併資料

list::remove

- 範例:

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

int main ()
{
  int a[]= {17,89,7,14};
  list<int> L (a,a+4);
  list<int>::iterator it;

  L.remove(89);

  for (it=L.begin(); it! =L.end(); ++it)
    cout << " " << *it;
  cout << endl;

  return 0;
}
```

list::reverse

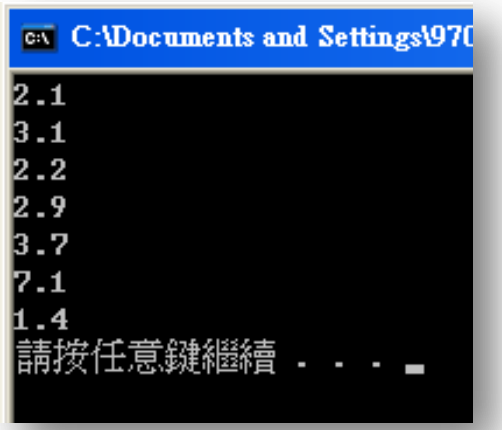
- 範例:

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

int main ()
{
 list<int> L;
 list<int>::iterator it;

 for (int i=1; i<10; i++)
                L.push_back(i);

 for (it=L.begin(); it! =L.end(); ++it)
  cout << " " << *it;
 cout << endl;

 L.reverse(); //反轉串列

 for (it=L.begin(); it! =L.end(); ++it)
  cout << " " << *it;
 cout << endl;

 return 0;
}
```

list::merge

- 範例:

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

int main ()
{
  list<double> L1, L2;
  list<double>::iterator it;

  L1.push_back (3.1);
  L1.push_back (2.2);
  L1.push_back (2.9);

  L2.push_back (3.7);
  L2.push_back (7.1);
  L2.push_back (1.4);
  L1.merge(L2);

  L2.push_back (2.1);
  L1.merge(L2);

  for (it=L1.begin(); it! =L1.end(); ++it)
    cout << *it << endl;

  return 0;
}
```



請寫一段程式碼看看L2此時的內容

映射表 map

- map是一種關聯容器，支援查表功能之資料結構，資料均以key/value方式存入

常用函式	
[n]	取得key值為n之value
size	計算長度
find	尋找資料
erase	刪除資料
insert	插入資料
count	判斷資料是否存在
lower_bound, (upper_bound)	取得接近下限(上限)之資料

map::map

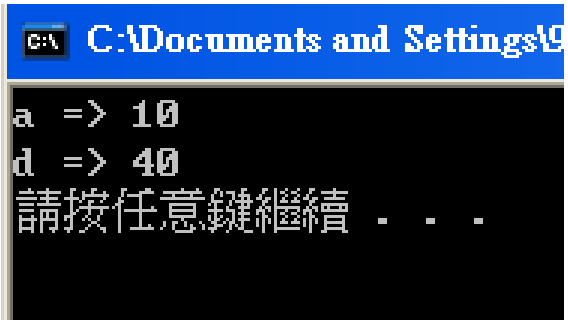
- 範例:

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

int main ()
{
  map<char,string> m;
  map<char,string>::iterator it;

  m['a']="Andy Lee";
  m['b']="Joe Wang";
  m['c']="Mary Lin";

  for (it=m.begin();it! =m.end();it++) {
     cout << "m['" << (*it).first << "'] is " << (*it).second << endl;
  }

  cout << "m now contains " << m.size() << " elements." << endl;

  return 0;
}
```

map::find, erase

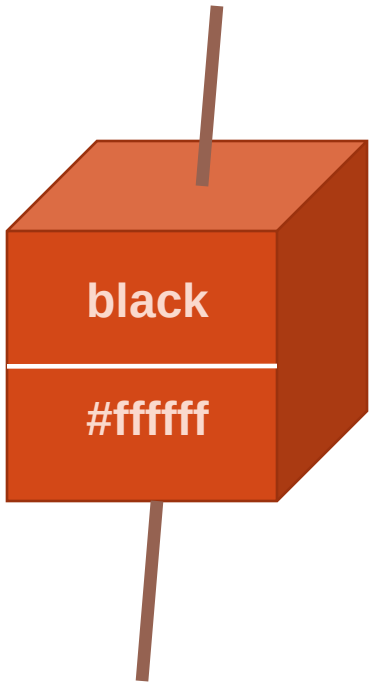
- 範例:

```
// erasing from map
#include <iostream>
#include <map>
using namespace std;

int main ()
{
  map<char,int> m;
  map<char,int>::iterator it;

  // insert some values:
  m['a']=10;
  m['b']=20;
  m['c']=30;
  m['d']=40;
  m['e']=50;
  m['f']=60;

  it=m.find('b');
  m.erase (it);                  // 刪除it位置之資料
  m.erase ('c');                 // 刪除key為'c'之資料
  it=m.find ('e');
  m.erase ( it, m.end() );    // 刪除一段資料

  for ( it=m.begin() ; it != m.end(); it++ )
    cout << (*it).first << " => " << (*it).second << endl;

  return 0;
}
```



map::insert

- 範例:



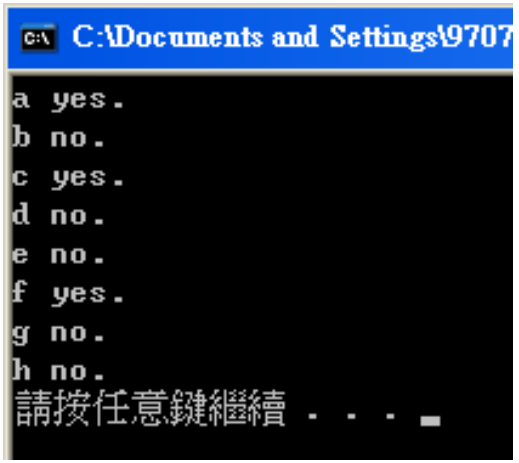

```
#include <iostream>
#include <iomanip>
#include <string>
#include <map>
using namespace std;

int main() {
   string s[][2] = {
      {"black",   "#ffffff"},
      {"white",   "#000000"},
      {"blue",    "#0000ff"},
      {"red",     "#ff0000"},
      {"green",   "#00ff00"},
      {"", ""}
   };
   map<string, string> m;
   map<string, string>::iterator it;
   for (int i = 0; !s[i][0].empty(); ++i)
      m.insert(map<string,string>::value_type(s[i][0], s[i][1]));

   for (it = m.begin(); it != m.end(); ++it)
   {
      cout << (*it).first   << " => " << (*it).second << endl;
   }
   if ((it = m.find("blue")) != m.end())
   {
      cout << "'blue' Find! It's value is " << (*it).second << endl;
   }
   return 0;
}
```

```
C:\Documents and Settings\970722\桌面\1.exe
black => #ffffff
blue => #0000ff
green => #00ff00
red => #ff0000
white => #000000
'blue' Find! It's value is #0000ff
請按任意鍵繼續 . . .
```

map::count

- 範例:

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

int main ()
{
  map<char,int> m;
  char c;

  m ['a']=101;
  m ['c']=202;
  m ['f']=303;

  for (c='a'; c<='h'; c++)
  {
    cout << c;
    if (m.count(c)>0)
      cout << " yes.\n";
    else
      cout << " no.\n";
  }

  return 0;
}
```



map::lower_bound, upper_bound

- 範例:

```
// map::lower_bound/upper_bound
#include <iostream>
#include <map>
using namespace std;

int main ()
{
 map<char,int> m;
 map<char,int>::iterator it,itlow,itup;

 m['a']=20;
 m['c']=40;
 m['d']=60;
 m['g']=80;
 m['i']=100;

 itlow=m.lower_bound ('b');  // 設定下限
 itup=m.upper_bound ('d');   // 設定上限

 m.erase(itlow,itup);       // 刪除下限到上限所有資料

 // print content:
 for ( it=m.begin() ; it ! = m.end(); it++ )
   cout << (*it).first << " => " << (*it).second << endl;

 return 0;
}
```

```
C:\Documents and Settings\9707
a => 20
g => 80
i => 100
請按任意鍵繼續 . . .
```

upper_bound():
回傳一個 iterator pointing 指到第一個container中key比input大的元素，相等時不回傳

lower_bound():
回傳一個 iterator pointing 指到第一個container中key不比input小的元素，相等時會回傳

大綱

- 標準樣板函式庫 (STL)
  - 抽象指標 (Iterator)
  - 容器 (Container)
  - 演算法 (Algorithm)

演算法(Algorithm)

- STL中的演算法
- 在STL中除了提供容器(類別樣板), 尚提供演算法(函式樣板)以供操作之資料
- #include <algorithm>
- 常用演算法
  - find
  - count
  - search
  - merge
  - sort
  - for_each
  - transform

find

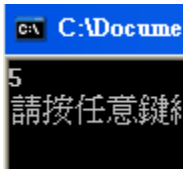
- find(first, last, val)
  - [first]: iterator
  - [last]: iterator
  - [val]: target value type
  - [回傳值]: iterator
- 針對某個container做搜尋，區間由first及last這兩個iterator指定，目標值為val，找到後回傳其所在元素指標(也是以iterator表示)
- 函數模板原型

```
template<class InIt, class T>
InIt find(InIt first, InIt last, const T& val);
```

find

- 範例

```
#include<iostream>
#include<algorithm>
using namespace std;
int main( )
{
   int l[7] = { 1, 3, 2, 5, 1, 2, 1 };
   int *it ;
   it = find(&l[0], &l[7], 5);
   if ( it == l+7)
      cout << "data not found\n" ;
   else
      cout << *it << endl;
   return 0;
}
```




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

int main( )
{
   list<int> L;
   list<int>::iterator it;
   L.push_back(10) ;
   L.push_back(20);
   L.push_back(30);
   it = find(L.begin( ), L.end( ), 30);
   if ( it == L.end( ))
      cout << "data not found\n";
   else
      cout << *it << endl;
   return 0;
}
```

count

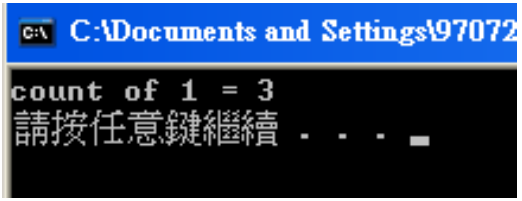
- count(first, last, val)
  - [first]: iterator
  - [last]: iterator
  - [val]: target value type
  - [回傳值]: int
- 針對某個container做搜尋，區間由first及last這兩個iterator指定，目標值為val，回傳container中元素值為val的個數
- 函數模板原型

```
template<class InIt, class T>
int count(InIt first, InIt last, const T& val);
```

count

- 範例

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

int main()
{
   int a[7] = { 1, 3, 2, 4, 1, 2, 1};
   vector<int> v(a,a+7);
   int count_of_1;
   count_of_1 = count(v.begin(), v.end(), 1);
   cout << "count of 1 = " <<count_of_1<<endl;

   return 0;
}
```



search

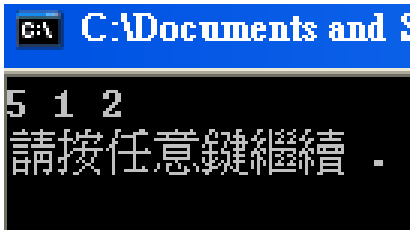
- search(s_first, s_last, t_first, t_last)
  - [s_first][t_first]: iterator
  - [s_last][t_last]: iterator
  - [回傳值]: iterator
- 找尋某一資料序列是否出現在另一個容器中
- 函數模板原型

```
template<class FwdIt1, class FwdIt2>
FwdIt1 search(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2);
```

search

- 範例:在STL的list容器中做搜尋

```
#include<iostream>
#include<list>
#include<vector>
#include<algorithm>
using namespace std;
int main( )
{
   int a[7] = { 1, 3, 2, 5, 1, 2, 1};
   vector<int> v(a,a+7) ;
   vector<int>::iterator it ;
   list<int> L ;
   L.push_back(5) ; L.push_back(1); L.push_back(2) ;
   it = search(v.begin( ), v.end( ), L.begin( ), L.end( ));
   if (it != v.end( )) //有找到
      cout << *it << " " << *(it+1) << " " << *(it+2) << endl ;

   return 0;
}
```



merge

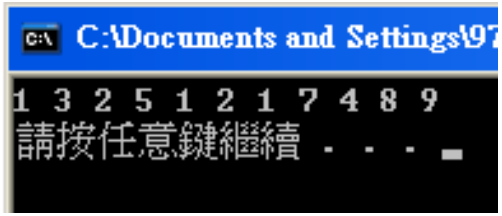
- merge(s1_first, s1_last, s2_first, s2_last, t_first)
  - [s1_first][s2_first]: iterator
  - [s1_last][s2_last]: iterator
  - [t_first]: iterator
  - [回傳值]: iterator
- 合併s1與s2兩資料於t
- 函數模板原型

```
template<class InIt1, class InIt2, class OutIt>
OutIt merge(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2, OutIt x);
```

merge

- 範例:

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
int main( )
{
   int a[7] = { 3, 2, 5, 1, 2, 1, 8 } ;
   int b[4] = { 1, 7, 4, 9 } ;
   vector<int> v1(a,a+7) ;
   vector<int> v2(b,b+4) ;
   vector<int> v3(v1.size( ) + v2.size( )) ;
   merge(v1.begin( ),v1.end( ),v2.begin( ),v2.end( ), v3.begin( ));
   for (int i=0 ; i<v3.size( ) ; i++ )
      cout << v3[i] << " " ;
   cout << endl;
   return 0;
}
```



這個結果是怎麼來的？

sort

- sort(first, last)
- sort(first, last, f)
  - [first]: iterator
  - [last]: iterator
  - [f]: 函式
  - [回傳值]: void
- 資料排序 (預設由小到大)
- 函數模板原型

```
template<class RanIt>
void sort(RanIt first, RanIt last);

template<class RanIt, class Pred>
void sort(RanIt first, RanIt last, Pred pr);
```

sort

- 範例:

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

bool  mygreater(int x, int y)    ←改成函數樣板試試看
{
   return x>y;
}

int main()
{
   int a[7] = { 8, 1, 3, 2, 5, 1, 4};
   vector<int> v(a,a+7);
   vector<int>::iterator it ;

   sort(v.begin( ), v.end( )) ;  //由小排到大
   for ( it = v.begin( ); it != v.end( ) ; it++)
      cout << *it <<" ";
   cout << endl;

   sort(v.begin( ), v.end( ), mygreater) ;  //mygreater 由大排到小
   for ( it = v.begin( ); it != v.end( ) ; it++)
      cout << *it <<" ";
   cout << endl;

   return 0;
}
```

for_each

- for_each(first, last, f)
  - [first]: iterator
  - [last]: iterator
  - [f]: 函式
  - [回傳值]: 函數物件指標
- 對container中從first 所指的元素起到last為止，其間每一個元素做某個動作(由函數f指定)
- 函數模板原型

```
template<class InIt, class Fun>
Fun for_each(InIt first, InIt last, Fun f) ;
```

for_each

- 範例:

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

void print(int &x)
{
   cout << x << " ";
}

void add(int &x)
{
   x = x+5;
}

int main( )
{
   list<int> L ;
   L.push_back(10);
   L.push_back(20) ;
   L.push_back(30) ;
   for_each(L.begin( ), L.end( ), print) ;
   cout << endl;
   for_each(L.begin( ), L.end( ), add) ;
   for_each(L.begin( ), L.end( ), print) ;
   cout << endl;
   return 0;
}
```

transform

- transform(first, last, output, f)
  - [first]: iterator
  - [last]: iterator
  - [output]: container
  - [f]: 函式
  - [回傳值]: 函數物件指標
- 同for_each，但會把結果放在output容器中
- 函數模板原型

```
template<class InIt, class OutIt, class Unop>
OutIt transform(InIt first, InIt last, OutIt x, Unop uop);
```

transform

- 範例:

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

void print(int &x)
{
   cout << x << " ";
}

int add(int x)
{
   return x+5;
}

int main( )
{
   list<int> L ;
   vector<int> v(3) ;
   L.push_back(10);
   L.push_back(20) ;
   L.push_back(30) ;
   transform(L.begin( ), L.end( ), v.begin(), add) ;
   for_each(L.begin( ), L.end( ), print) ;
   cout << endl;
   for_each(v.begin( ), v.end( ), print) ;
   cout << endl;
   return 0;
}
```

STL參考網站

- http://www.cplusplus.com/reference/stl/