



# Lecture 11

## Thư viện chuẩn

## Standard Template Library

# Thư viện Standard Template Library (STL)

- STL
  - 3 thành phần chính
    - Containers: Các cấu trúc dữ liệu template
    - Iterators: giống con trỏ, dùng để truy cập vào các phần tử của containers
    - Algorithms: thao tác, tìm kiếm, sắp xếp... dữ liệu.
  - Object- oriented programming: TÁI SỬ DỤNG
  - STL là 1 thư viện class không lỗi

## 21.1.1 Giới thiệu về Containers

- Có 3 loại containers
  - Containers tuần tự
    - Cấu trúc dữ liệu tuyến tính (vectors, danh sách liên kết linked lists)
    - First-class container
  - Associative containers
    - Không tuyến tính, có thể tìm các phần tử nhanh
    - Cặp khoá/giá trị
    - First-class container
  - Container adapters
    - Tương tự như containers, nhưng ít tính năng hơn
- Containers có một số functions chung

## Các class của STL Container (Fig. 21.1)

- Containers tuần tự
  - vector
  - deque
  - list
- Associative containers
  - set
  - multiset
  - map
  - multimap
- Container adapters
  - stack
  - queue
  - priority\_queue

# Các hàm thành viên thông dụng của STL (Fig. 21.2)

- Các hàm thành viên của tất cả containers
  - Hàm tạo mặc định, hàm tạo copy, hàm huỷ
  - `empty`
  - `max_size`, `size`
  - `=` `<` `<=` `>` `>=` `==` `!=`
  - `swap`
- Các hàm cho first-class containers
  - `begin`, `end`
  - `rbegin`, `rend`
  - `erase`, `clear`

## Các kiểu thông dụng của STL (Fig. 21.4)

- typedefs cho first-class containers
  - value\_type
  - reference
  - const\_reference
  - pointer
  - iterator
  - const\_iterator
  - reverse\_iterator
  - const\_reverse\_iterator
  - difference\_type
  - size\_type

## 21.1.2 Giới thiệu về Iterators

- Iterators tương tự pointers
  - Trỏ tới phần tử đầu tiên trong 1 container
  - Các phép toán (operators) iterator cho toàn bộ containers là giống nhau
    - \* tham chiếu
    - ++ trỏ tới phần tử kế
    - `begin()` trả về iterator cho phần tử đầu
    - `end()` trả về iterator cho phần tử cuối
  - Sử dụng iterators với sequences (ranges)
    - Containers
    - Input sequences: `istream_iterator`
    - Output sequences: `ostream_iterator`

## 21.1.2 Giới thiệu về Iterators

- Cách dùng

- `std::istream_iterator< int > inputInt( cin )`
  - Đọc input từ `cin`
  - `*inputInt`
    - Tham chiếu tới int đầu tiên từ `cin`
  - `++inputInt`
    - Tới int tiếp từ luồng (stream)
- `std::ostream_iterator< int > outputInt(cout)`
  - Đưa ints tới `cout`
  - `*outputInt = 7`
    - Đưa 7 tới `cout`
  - `++outputInt`
    - Đưa int tiếp





```
1  // Fig. 21.5: fig21_05.cpp
2  // Demonstrating input and output with iterators.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iterator> // ostream_iterator and
10 ostream_iterator
11
12 int main()
13 {
14     cout << "Enter two integers: ";
15
16     // create ostream_iterator for reading int value
17     cin
18     std::istream_iterator< int > inputInt( cin );
19
20     int number1 = *inputInt; // read int from standard input
21     ++inputInt;             // move iterator to next input value
22     int number2 = *inputInt; // read int from standard input
23 }
```

Truy cập và gán iterator như 1  
con trỏ



fig21\_05.cpp

(2 of 2)

fig21\_05.cpp

output (1 of 1)

```
22 // create ostream_iterator for writing int values to
cout
23 std::ostream_iterator< int > outputInt( cout );
24
25 cout << "The sum is: ";
26 *outputInt = number1 + number2; // output result to
cout
27 cout << endl;
28
29 return 0;
30
31 } // end main
```

Tương tự, tạo 1  
ostream\_iterator. Gán  
iterator outputs này cho  
cout.

```
Enter two integers: 12 25
The sum is: 37
```

## Iterator Categories (Fig. 21.6)

- Input
  - Đọc các phần tử từ container, chỉ có thể di chuyển tiến lên (forward)
- Output
  - Viết các phần tử vào container, chỉ tiến lên (forward)
- Forward
  - Kết hợp cả input và output, nhưng giữ nguyên vị trí
  - Nhiều lần duyệt (có thể duyệt 1 chuỗi 2 lần)
- 2 chiều
  - Giống forward, nhưng có thể di chuyển cả tiến và lùi
- Truy cập ngẫu nhiên
  - Giống 2 chiều, nhưng có thể nhảy tới bất kì vị trí nào

## Iterator Types Supported (Fig. 21.8)

- **Sequence containers**
  - `vector`: truy cập ngẫu nhiên
  - `deque`: truy cập ngẫu nhiên
  - `list`: 2 hướng
- **Associative containers (2 chiều)**
  - `set`
  - `multiset`
  - `Map`
  - `multimap`
- **Container adapters (không hỗ trợ iterators)**
  - `stack`
  - `queue`
  - `priority_queue`

## Iterator Operations (Fig. 21.10)

- Tất cả
  - $++p, p++$
- Input iterators
  - $*p$
  - $p = p1$
  - $p == p1, p \neq p1$
- Output iterators
  - $*p$
  - $p = p1$
- Forward iterators
  - Có chức năng cho vào/ra iterators

## Iterator Operations (Fig. 21.10)

- 2 chiều
  - `--p, p--`
- Truy cập ngẫu nhiên
  - `p + i, p += i`
  - `p - i, p -= i`
  - `p[i]`
  - `p < p1, p <= p1`
  - `p > p1, p >= p1`

## 21.1.3 Giới thiệu về thuật toán

- STL có các thuật toán được dùng trong các containers
  - Thao tác trên các phần tử gián tiếp thông qua iterators
  - Thường thao tác trên các chuỗi phần tử
    - Được định nghĩa bởi 1 cặp iterators
    - Phần tử đầu và cuối
  - Các thuật toán thường trả về iterators
    - `find()`
    - Trả về các iterator cho các phần tử, hoặc `end()` nếu không tìm thấy
  - Các thuật toán (đã viết sẵn) giúp lập trình viên tiết kiệm thời gian và công sức

## 21.2 Sequence Containers

- 3 sequence containers
  - `vector` – dựa trên array
  - `deque` – dựa trên array
  - `list` – danh sách liên kết mạnh



## 21.2.1 vector Sequence Container

- vector
  - `<vector>`
  - Cấu trúc dữ liệu với các địa chỉ nhớ liên kề nhau
    - Truy cập các phần tử bằng `[]`
  - Sử dụng khi dữ liệu cần được sắp xếp và truy cập đơn giản
- Khi hết bộ nhớ
  - Cấp phát một vùng nhớ liên tiếp lớn hơn
  - Copy dữ liệu tới vùng nhớ mới
  - Huỷ bỏ vùng nhớ cũ
- Có iterators truy cập ngẫu nhiên

## 21.2.1 vector Sequence Container

- Khai báo

- `std::vector <type> v;`
  - `type`: `int`, `float`, etc.

- Iterators

- `std::vector<type>::const_iterator iterVar;`
  - `const_iterator` không thể thay đổi các phần tử
- `std::vector<type>::reverse_iterator iterVar;`
  - Duyệt các phần tử theo thứ tự ngược (từ cuối lên đầu)
  - Dùng `rbegin` để lấy điểm bắt đầu
  - Dùng `rend` để lấy điểm cuối cùng

## 21.2.1 vector Sequence Container

- **vector functions**

- `v.push_back(value)`
  - Thêm phần tử vào cuối (có trong tất cả sequence containers).
- `v.size()`
  - Kích thước hiện tại của vector
- `v.capacity()`
  - Khả năng lưu hiện tại của vector trước khi phải cấp vùng nhớ mới
- `vector<type> v(a, a + SIZE)`
  - Tạo vector `v` với các phần tử từ array `a` tới kích thước `a + SIZE`

## 21.2.1 vector Sequence Container

- vector functions
  - `v.insert(iterator, value)`
    - Chèn *value* trước vị trí của *iterator*
  - `v.insert(iterator, array, array + SIZE)`
    - Chèn phần tử array (tới trước *array + SIZE*) vào vector
  - `v.erase(iterator)`
    - Xoá phần tử khỏi container
  - `v.erase(iter1, iter2)`
    - Xoá các phần tử từ *iter1* tới trước *iter2*
  - `v.clear()`
    - Xoá toàn bộ container

## 21.2.1 vector Sequence Container

- vector functions operations
  - `v.front()`, `v.back()`
    - Trả về phần tử đầu và cuối
  - `v[elementNumber] = value;`
    - Gán value cho 1 phần tử
  - `v.at[elementNumber] = value;`
    - Giống trên, có kiểm tra khoảng
    - ngoại lệ `out_of_bounds`

## 21.2.1 vector Sequence Container

- ostream\_iterator

- `std::ostream_iterator< type >`  
`Name( outputStream, separator );`

- *type*: giá trị outputs của kiểu hiện tại
    - *outputStream*: vị trí iterator output
    - *separator*: kí tự ngăn cách các outputs

- Ví dụ

- `std::ostream_iterator< int > output( cout, " " );`
  - `std::copy( iterator1, iterator2, output );`

- Copy các phần tử từ *iterator1* tới *iterator2* ra *output*, là một `ostream_iterator`



fig21\_14.cpp  
(1 of 3)

```
1 // Fig. 21.14: fig21_14.cpp
2 // Demonstrating standard library vector class template.
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 #include <vector> // vector class-template definition
10
11 // prototype for function template printVector
12 template < class T >
13 void printVector( const std::vector< T > &integers2 );
14
15 int main()
16 {
17     const int SIZE = 6;
18     int array[ SIZE ] = { 1, 2, 3, 4, 5, 6 };
19
20     std::vector< int > integers;
21
22     cout << "The initial size of integers is: "
23         << integers.size()
24         << "\nThe initial capacity of integers is: "
25         << integers.capacity();
26 }
```

Tạo 1 vector kiểu ints.

Gọi các hàm thành viên



Thêm các phần tử vào cuối  
vector dùng push\_back.

fig21\_14.cpp  
(2 of 3)

```
27 // function push_back is in every sequence
28 integers.push_back( 2 );
29 integers.push_back( 3 );
30 integers.push_back( 4 );
31
32 cout << "\nThe size of integers is: " <<
integers.size()
33     << "\nThe capacity of integers is: "
34     << integers.capacity();
35
36 cout << "\n\nOutput array using pointer notation: ";
37
38 for ( int *ptr = array; ptr != array + SIZE; ++ptr )
39     cout << *ptr << ' ';
40
41 cout << "\n\nOutput vector using iterator notation: ";
42 printVector( integers );
43
44 cout << "\n\nReversed contents of vector integers: ";
45
```





```
46  std::vector< int >::reverse_iterator reverseIterator;
```

```
47  
48  for ( reverseIterator = integers.rbegin();
```

```
49      reverseIterator != integers.rend();
```

```
50      ++reverseIterator )
```

```
51      cout << *reverseIterator << ' ';
```

```
52  
53  cout << endl;
```

```
54  
55  return 0;
```

```
56  
57 } // end main
```

```
58  
59 // function template for outputting vector el
```

```
60 template < class T >
```

```
61 void printVector( const std::vector< T > &integers2 )
```

```
62 {
```

```
63     std::vector< T >::const_iterator constIterator;
```

```
64  
65     for ( constIterator = integers2.begin();
```

```
66         constIterator != integers2.end();
```

```
67         constIterator++ )
```

```
68         cout << *constIterator << ' ';
```

```
69  
70 } // end function printVector
```

Duyệt vector theo chiều  
ngược sử dụng  
reverse\_iterator.

Template function để duyệt  
qua vector thuận.



fig21\_14.cpp  
output (1 of 1)

The initial size of v is: 0

The initial capacity of v is: 0

The size of v is: 3

The capacity of v is: 4

Contents of array a using pointer notation: 1 2 3 4 5 6

Contents of vector v using iterator notation: 2 3 4

Reversed contents of vector v: 4 3 2

## 21.3 Associative Containers

- Associative containers
  - Truy cập trực tiếp để lưu trữ/đọc các phần tử
  - Sử dụng khoá (search keys)
  - 4 loại: `multiset`, `set`, `multimap` and `map`
    - Các khoá theo thứ tự sắp xếp
    - `multiset` and `multimap` cho phép khoá trùng nhau
    - `multimap` and `map` có các khoá và các giá trị liên quan
    - `multiset` and `set` chỉ có các giá trị

## 21.3.1 multiset Associative Container

- `multiset`
  - Header `<set>`
  - Lưu trữ, lấy key nhanh
  - Cho phép trùng lặp
  - Iterators 2 chiều
- Sắp xếp các phần tử
  - Thực hiện bằng hàm so sánh đối tượng
    - Dùng khi tạo `multiset`
  - Với `integer multiset`
    - `multiset< int, std::less<int> > myObject;`
    - Các phần tử sẽ được lưu trữ theo giá trị tăng dần

## 21.3.1 multiset Associative Container

- Multiset functions

- `ms.insert(value)`
  - Chèn *value* vào multiset
- `ms.count(value)`
  - Số lần xuất hiện *value*
- `ms.find(value)`
  - Trả về iterator cho phần tử đầu có *value*
- `ms.lower_bound(value)`
  - Trả về iterator cho vị trí đầu có *value*
- `ms.upper_bound(value)`
  - Trả về iterator cho vị trí tiếp của phần tử cuối cùng có *value*

## 21.3.1 multiset Associative Container

- Class `pair`
  - Thao tác trên các cặp giá trị
  - `Pair objects` chứa `first` và `second`
    - `const_iterators`
  - Cho 1 `pair` đối tượng `q`
    - `q = ms.equal_range(value)`
    - Thiết lập `first` và `second` bằng `lower_bound` và `upper_bound` bằng `value`



```
1 // Fig. 21.19: fig21_19.cpp
2 // Testing Standard Library class multiset
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <set> // multiset class-template definition
9
10 // define short name for multiset type used in this program
11 typedef std::multiset< int, std::less< int > > ims;
12
13 #include <algorithm> // copy algorithm
14
15 int main()
16 {
17     const int SIZE = 10;
18     int a[ SIZE ] = { 7, 22, 9, 1, 18, 30, 100, 22, 85, 13 };
19
20     ims intMultiset; // ims is typedef for "integer
multiset"
21     std::ostream_iterator< int > output( cout, " " );
22
23     cout << "There are currently " << intMultiset.count( 15 )
24         << " values of 15 in the multiset\n";
25
```

Dùng typedefs giúp chương trình sáng sủa hơn. Định nghĩa một multiset integers để lưu giá trị theo thứ tự tăng dần.



fig21\_19.cpp  
(2 of 3)

```
26  intMultiset.insert( 15 ); // insert 15 in intMultiset
27  intMultiset.insert( 15 ); // insert 15 in intMultiset
28
29  cout << "After inserts, there are "
30        << intMultiset.count( 15 )
31        << " values of 15 in the multiset\n\n";
32
33  // iterator that cannot be used to change elements
34  ims::const_iterator result;
35
36  // find 15 in intMultiset; find returns iterator
37  result = intMultiset.find( 15 );
38
39  if ( result != intMultiset.end() ) // if iterator not at end
40      cout << "Found value 15\n";    // found search value 15
41
42  // find 20 in intMultiset; find returns iterator
43  result = intMultiset.find( 20 );
44
45  if ( result == intMultiset.end() ) // will be true hence
46      cout << "Did not find value 20\n"; // did not find 20
47
48  // insert elements of array a into intMultiset
49  intMultiset.insert( a, a + SIZE );
50
51  cout << "\nAfter insert, intMultiset contains:\n";
52  std::copy( intMultiset.begin(), intMultiset.end(), output );
53
```

Sử dụng hàm thành viên  
find.





fig21\_19.cpp  
(3 of 3)

```
54 // determine lower and upper bound of 22 in intMultiset
55 cout << "\n\nLower bound of 22: "
56     << *( intMultiset.lower_bound( 22 ) );
57 cout << "\nUpper bound of 22: "
58     << *( intMultiset.upper_bound( 22 ) );
59
60 // p represents pair of const_iterators
61 std::pair< ims::const_iterator, ims::const_iterator > p;
62
63 // use equal_range to determine lower and upper bounds
64 // of 22 in intMultiset
65 p = intMultiset.equal_range( 22 );
66
67 cout << "\n\nequal_range of 22:"
68     << "\n    Lower bound: " << *( p.first )
69     << "\n    Upper bound: " << *( p.second );
70
71 cout << endl;
72
73 return 0;
74
75 } // end main
```

Sử dụng đối tượng pair để lấy cận trên và cận dưới của 22.



There are currently 0 values of 15 in the multiset

After inserts, there are 2 values of 15 in the multiset

Found value 15

Did not find value 20

After insert, intMultiset contains:

1 7 9 13 15 15 18 22 22 30 85 100

Lower bound of 22: 22

Upper bound of 22: 30

equal\_range of 22:

Lower bound: 22

Upper bound: 30

## 21.4 Container Adapters

- Container adapters
  - `stack`, `queue` and `priority_queue`
  - Không phải là first class containers
    - Không hỗ trợ iterators
    - Không cung cấp cấu trúc dữ liệu cụ thể
  - Hàm thành viên `push` and `pop`

## 21.4.1 stack Adapter

- stack
  - Header `<stack>`
  - Chèn và xoá ở 1 đầu
  - Cấu trúc dữ liệu: Last-in, first-out (LIFO)
  - Có thể dùng `vector`, `list`, or `deque` (mặc định)
  - Định nghĩa

```
stack<type, vector<type> > myStack;
stack<type, list<type> > myOtherStack;
stack<type> anotherStack; // default deque
```

    - `vector`, `list`
      - Cài đặt của `stack` (mặc định `deque`)



fig21\_23.cpp  
(1 of 3)

```
1  // Fig. 21.23: fig21_23.cpp
2  // Standard library adapter stack test program.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <stack>    // stack adapter definition
9  #include <vector>   // vector class-template definition
10 #include <list>     // list class-template definition
11
12 // popElements function-template prototype
13 template< class T >
14 void popElements( T &stackRef );
15
16 int main()
17 {
18     // stack with default underlying deque
19     std::stack< int > intDequeStack;
20
21     // stack with underlying vector
22     std::stack< int, std::vector< int > > intVectorStack;
23
24     // stack with underlying list
25     std::stack< int, std::list< int > > intListStack;
26
```

Tạo các stack với các cài đặt khác nhau.



Sử dụng hàm thành viên  
push.

21\_23.cpp  
of 3)

```
27 // push the values 0-9 onto each stack
28 for ( int i = 0; i < 10; ++i ) {
29     intDequeStack.push( i );
30     intVectorStack.push( i );
31     intListStack.push( i );
32
33 } // end for
34
35 // display and remove elements from each stack
36 cout << "Popping from intDequeStack: ";
37 popElements( intDequeStack );
38 cout << "\nPopping from intVectorStack: ";
39 popElements( intVectorStack );
40 cout << "\nPopping from intListStack: ";
41 popElements( intListStack );
42
43 cout << endl;
44
45 return 0;
46
47 } // end main
48
```



fig21\_23.cpp

(3 of 3)

fig21\_23.cpp

output (1 of 1)

```
49 // pop elements from stack object to which stackRef
    refers
50 template< class T >
51 void popElements( T &stackRef )
52 {
53     while ( !stackRef.empty() ) {
54         cout << stackRef.top() << ' '; // view top
    element
55         stackRef.pop();                // remove top
    element
56
57     } // end while
58
59 } // end function popElements
```

```
Popping from intDequeStack: 9 8 7 6 5 4 3 2 1 0
Popping from intVectorStack: 9 8 7 6 5 4 3 2 1 0
Popping from intListStack: 9 8 7 6 5 4 3 2 1 0
```

## 21.5 Algorithms

- STL phân tách containers và thuật toán
  - Dễ thêm các thuật toán mới
  - Hiệu quả, tránh gọi hàm ảo
  - `<algorithm>`



## 21.5.6 Các thuật toán sắp xếp và tìm kiếm cơ bản

- `find(iter1, iter2, value)`
  - Trả về iterator của phần tử đầu tiên bằng `value` (trong khoảng)
- `find_if(iter1, iter2, function)`
  - Giống `find`
  - Trả về iterator khi `function` bằng `true`
- `sort(iter1, iter2)`
  - Sắp xếp theo thứ tự tăng dần
- `binary_search(iter1, iter2, value)`
  - Tìm kiếm trên danh sách đã sắp xếp tăng dần
  - Sử dụng tìm kiếm nhị phân



fig21\_31.cpp  
(1 of 4)

```
1 // Fig. 21.31: fig21_31.cpp
2 // Standard library search and sort algorithms.
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 #include <algorithm> // algorithm definitions
9 #include <vector>    // vector class-template definition
10
11 bool greater10( int value ); // prototype
12
13 int main()
14 {
15     const int SIZE = 10;
16     int a[ SIZE ] = { 10, 2, 17, 5, 16, 8, 13, 11, 20, 7 };
17
18     std::vector< int > v( a, a + SIZE );
19     std::ostream_iterator< int > output( cout, " " );
20
21     cout << "Vector v contains: ";
22     std::copy( v.begin(), v.end(), output );
23
24     // locate first occurrence of 16 in v
25     std::vector< int >::iterator location;
26     location = std::find( v.begin(), v.end(), 16 );
```



fig21\_31.cpp  
(2 of 4)

```
27
28 if ( location != v.end() )
29     cout << "\n\nFound 16 at location "
30         << ( location - v.begin() );
31 else
32     cout << "\n\n16 not found";
33
34 // locate first occurrence of 100 in v
35 location = std::find( v.begin(), v.end(), 100 );
36
37 if ( location != v.end() )
38     cout << "\n\nFound 100 at location "
39         << ( location - v.begin() );
40 else
41     cout << "\n\n100 not found";
42
43 // locate first occurrence of value greater than 10 in v
44 location = std::find_if( v.begin(), v.end(), greater10 );
45
46 if ( location != v.end() )
47     cout << "\n\nThe first value greater than 10 is "
48         << *location << "\n\nfound at location "
49         << ( location - v.begin() );
50 else
51     cout << "\n\nNo values greater than 10 were found";
52
```



fig21\_31.cpp

(3 of 4)

```
53 // sort elements of v
54 std::sort( v.begin(), v.end() );
55
56 cout << "\n\nVector v after sort: ";
57 std::copy( v.begin(), v.end(), output );
58
59 // use binary_search to locate 13 in v
60 if ( std::binary_search( v.begin(), v.end(), 13 ) )
61     cout << "\n\n13 was found in v";
62 else
63     cout << "\n\n13 was not found in v";
64
65 // use binary_search to locate 100 in v
66 if ( std::binary_search( v.begin(), v.end(), 100 ) )
67     cout << "\n\n100 was found in v";
68 else
69     cout << "\n\n100 was not found in v";
70
71 cout << endl;
72
73 return 0;
74
75 } // end main
76
```



fig21\_31.cpp  
(4 of 4)

fig21\_31.cpp  
output (1 of 1)

```
77 // determine whether argument is greater than 10
78 bool greater10( int value )
79 {
80     return value > 10;
81
82 } // end function greater1
```

Vector v contains: 10 2 17 5 16 8 13 11 20 7

Found 16 at location 4

100 not found

The first value greater than 10 is 17

found at location 2

Vector v after sort: 2 5 7 8 10 11 13 16 17 20

13 was found in v

100 was not found in v