Mục lục

Contents

Chuyển đổi giữa chuỗi và số thông qua đối tượng stringstream (#include<sstream>)

› Giả sử ta cần đọc một dãy số nguyên từ bàn phím và cần in chúng ra màn hình: Ta có thể đọc cả dòng chứa các số đó vào một chuỗi và sử dụng stringstream để tách chuỗi lấy các số đó như sau:

```
string s; getline(cin,s);
stringstream ss(s);
int a;
while(ss>>a) cout<< a<<‘ ‘;
```

› Nếu việc tách số gặp lỗi thì ss.fail() == true

› Để tách chuỗi s = “Hoang Van Thu” thành các từ riêng biệt, ta cũng có thể làm tương tự như trên:

```
string s = “Hoang Van Thu”;
stringstream ss(s); // hoặc dùng lệnh ss.str(s);
string tu;
while(ss>>tu) cout<<tu<<‘ ‘;
```

› Giả sử ta cần đổi một số hoặc một dãy số thành một chuỗi số, ta cũng có thể sử dụng stringstream để thực hiện như sau:

```
int a = 12345, b = 678;
stringstream sstr; sstr << a << b;
string s; sstr >> s; // in ra 12345678
```

Xuất nhập với tệp văn bản (#include<fstream>)

A. Khai báo tệp văn bản input: ifstream fi(“dulieu.inp”);

B. Khai báo tệp văn bản output: ofstream fo(“ketqua.out”);

C. Đọc và ghi tệp văn bản:

```
fi >> x >> y >> z;
fo << x << ‘ ‘ << y << ‘ ‘ << z;
```

D. Đọc đến hết file: while(fi >> x) fo << x <<‘ ‘;

Maxlongint **** 1<<30**

Hằng bool: true (!=0); false (0);

if(0); // nếu sai

if(1); // nếu đúng

Lệnh lựa chọn switch

Cú pháp:

```
switch (biểu_thức_giá_trị){
    case hằng_số_1:
        nhóm_lệnh_1; break;
    case hằng_số_2:
        nhóm_lệnh_2; break;
    default:    nhóm_lệnh_mặc_định}
```

hàm đệ quy tính UCLN(a,b) và hàm tính lũy thừa a

```
int UCLN(int a, int b){ if(!b) return a; else return UCLN(b,a%b);}
int luythua(int a, int n){
    if(n==0) return 1;
    if(n==1) return a;
    int t = luythua(a,n/2);
    if (n % 2== 0) return t*t; else return t*t*a;
}
main(){
    cout<<UCLN(30,24)<<endl<<luythua(3,3);
}
```

Khai báo chồng phép toán

Tình huống: Khi ta so sánh hai điểm trên mặt phẳng theo tiêu chí ưu tiên hoành độ nhỏ hơn, nếu không thì ưu tiên tung độ nhỏ hơn.

Cách thực hiện 1: Định nghĩa hàm SOSANH(Point A, Point B)

```
bool Sosanh(Point P1, Point P2) {
    if(P1.x != P2.x) return P1.x < P2.x;
    else return P1.y < P2.y;
}
sort(a,a+100,Sosanh);
```

Cách thực hiện 2: Định nghĩa phép toán < dành riêng cho kiểu Point

```
struct Point{
    int x, y;
    bool operator < (const Point P) const{
        if(x != P.x) return x < P.x;
        else return y < P.y;
}
} a[100];
sort(a,a+100);
```

```
typedef pair<int, int> pii;
using piii = pair<pii, int>;
priority_queue<pii> heap;
```

String

Viết hoa

```
        int main ()
{
  int i=0;
  char str[]="Test String.\n";
  char c;
  while (str[i])
  {
    c=str[i];
    putchar (toupper(c));
    i++;
  }
  return 0;
} //TEST STRING.
```

Viết hoa: strupr(chuỗi)
Viết thường: strlwr(chuỗi)
Lấy mã trong ascii: int(s[i])

Chuyển từ mã ascii sang char

```
int main()
{
   for (int i=1;i<100;i++)
   {
      char c=i;
      cout<<c<<endl;
   }
   return 0;
}
```

Duyệt xâu

```
string::iterator it;
for(it = s.begin(); it != s.end(); it++) cout<< *it<<' ';
```

In ngược xâu bằng con trỏ duyệt ngược.

```
string::reverse_iterator it;
for (it = s.rbegin(); it != s.rend(); ++it) cout << *it;
```

Nhóm hàm và phép toán về kích thước	
Tên hàm	Ý nghĩa
size() / length()	trả về số lượng phần tử của string.
empty()	trả về true(1) nếu string rỗng, ngược lại là false(0)
clear()	xóa string
s[i]	trả về phần tử thứ i của biến string s. Không kiểm tra phạm vi
at(i)	trả về phần tử thứ i string s. Có kiểm tra phạm vi

Tên hàm	Ý nghĩa
toán tử +/+=	nối hai string. Ví dụ: s1 = s1+ s2 hoặc s1 += s2;
push_back (x)	thêm ký tự x vào cuối string
insert (i = 0, x)	chèn x vào trước vị trí i của string (x có thể là kiểu string/C string)
erase (i = 0, n =npos)	xóa *n* ký tự tính từ vị trí *i* của string
replace (i=0, n=npos, s)	Thay thế *n* ký tự tính từ vị trí *i* của string bằng *s* (*s* có thể là kiểu string/C string/char)

Tên hàm	Ý nghĩa
c_str ()	trả về chuỗi giá trị của string ở dạng C string (đổi từ str → C string)
copy (s, n, i=0)	trả về một chuỗi s dạng C-string là một phần của string, gồm n ký tự, tính từ vị trí i (mặc định i = 0). Hàm không tự động thêm '\0' vào cuối s.
find(s, i=0)	trả về vị trí đầu tiên xuất hiện chuỗi s trong string, tính từ vị trí i (mặc định i=0, s có thể là kiểu string/C string/char)
rfind(s, i=npos)	tương tự hàm find nhưng tìm từ cuối string trở lại, mặc định i là ký tự cuối chuỗi
substr(pos=0, len=npos)	trả về một xâu bằng cách copy xâu hiện tại từ vị trí pos (mặc định là từ vị trí 0) với độ dài len (mặc định là đến hết xâu)

STL C++

STL vector #include <vector>

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data C++11	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)

Phép duyệt vector

Cách 1: Duyệt như mảng tĩnh, ví dụ:

```
vector<int> v(10,100); // dãy 10 phần tử có giá trị 100
for(int i = 0; i < v.size(); i++) cout << v[i]<<' ';
for(int i = 0; i < v.size(); i++) cout <<v.at(i)<<' ';
```

Cách 2:

Duyệt với con trỏ lặp iterator tiến, ví dụ:

```
for(vector<int>::iterator it =v.begin(); it !=v.end(); it++)cout <<*it<<'';
```

Duyệt với con trỏ lặp iterator lùi, ví dụ:

```
int a[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
vector<int> v(a,a+10);
for(vector<int>::reverse_iterator it = v.rbegin();
it != v.rend(); it++) cout<<*it<<' ';
```

v.resize(10);

Chèn phần tử: O(sizeof(v))

```
vector<int> a(2,100);
```

```
vector<int>::iterator it;
it = a.begin();
it = a.insert (it, 200); //{200,100,100}
a.insert (it,2,300); //{300,300,100,100}
vector<int> b(2,400);
a.insert (it+2,b.begin(),b.end());
a={300,300, 400, 400, 100, 100}
int arr[] = { 501,502,503 };
a.insert (a.begin(), arr, arr+3);
a={501, 502, 503, 300, 300, 400, 400, 200, 100, 100}
```

xóa phần tử: O(sizeof(v))

```
vector<int> v;
for (int i=1; i<=10; i++) a.push_back(i);
// a={1, 2, 3, 4, 5, 7, 8, 9, 10}
myvector.erase (v.begin()+5); // xóa phần tử thứ 6
// a={1, 2, 3, 4, 5, 7, 8, 9, 10}
a.erase(a.begin(),a.begin()+3); // Xóa 3 phần tử đầu tiên
// a={4, 5, 7, 8, 9, 10}
a.pop_back(); // Xóa phần tử cuối
// a={4, 5, 7, 8, 9}
```

So sánh và hoán đổi giá trị 2 vector

```
int a[]= {11, 22, 33};
int b[]= {11, 22, 33, 4};
vector<int> v1 (a,a+3);
vector<int> v2 (b,b+4);
//so sánh giá trị 2 vector
if(v1==v2) cout<<"=\n"; //(empty)
if(v1<v2) cout<<"<\n"; // <
```

```
if(v1<=v2) cout<<"<=\n"; //<=
v1.swap(v2); //hoán đổi giá trị 2 vector
for(vector<int>::iterator it = v1.begin();
it!=v1.end(); it++) cout<<*it<<' ';
//11 22 33 4
```

STL list #include <list>

Element access:

front	Access first element (public member function)
back	Access last element (public member function)

Modifiers:

assign	Assign new content to container (public member function)
emplace_front C++11	Construct and insert element at beginning (public member function)
push_front	Insert element at beginning (public member function)
pop_front	Delete first element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
emplace C++11	Construct and insert element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
resize	Change size (public member function)
clear	Clear content (public member function)

Operations:

splice	Transfer elements from list to list (public member function)
remove	Remove elements with specific value (public member function)
remove_if	Remove elements fulfilling condition (public member function template)
unique	Remove duplicate values (public member function)
merge	Merge sorted lists (public member function)
sort	Sort elements in container (public member function)
reverse	Reverse the order of elements (public member function)

Observers:

get_allocator	Get allocator (public member function)

Khái niệm

• List là loại thùng chứa dãy cho phép ta thực hiện các thao tác chèn và xóa ở bất cứ vị trí nào trong dãy với thời gian hằng số, và lặp theo cả hai hướng.

• List phải trả giá là không thể truy xuất ngẫu nhiên phần tử bằng chỉ số như vector, array, deque. Ta phải duyệt từ đầu danh sách để tìm phần tử

Phép duyệt list và một số vị trí đặc biệt

```
int a[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

list<int> v(a,a+10);
```

• Duyệt với con trỏ lặp iterator tiến, ví dụ:

```
for(list<int>::iterator it = v.begin(); it != v.end(); it++) cout <<*it<<' ';
```

• Duyệt với con trỏ lặp iterator lùi, ví dụ:

```
for(list<int>::reverse_iterator it = v.rbegin();

it != v.rend(); it++) cout<<*it<<' ';
```

• Trả về giá trị phần tử đầu tiên trong list: v.front()

• Trả về giá trị phần tử cuối cùng trong list: v.back()

Nhập dãy vào list

```
list<int> v; // vector rỗng

int x;

for(int i = 0; i < 100; i++){

cin >> x; v.push_back(x);

}
```

Chèn phần tử vào list: O(1)

```
list<int> a; a<int>::iterator it;

for (int i=1; i<=5; ++i) a.push_back(i);

// 1 2 3 4 5

it = a.begin(); ++it;

// it trỏ đến phần tử thứ 2

a.insert(it,10); // 1 10 2 3 4 5

a.insert (it,2,20); // 1 10 20 20 2 3 4 5

--it; // it trỏ đến số thứ 4
```

```
vector<int> v(2,30);
a.insert (it,v.begin(),v.end());
// 1 10 20 30 30 20 2 3 4 5
```

• Chèn phần tử x vào đầu danh sách: a.push_front(x);

• Chèn phần tử x vào cuối danh sách: a.push_back(x)

xóa phần tử khỏi list: O(1)

```
list<int> a;
list<int>::iterator it1, it2;
for (int i=1; i<10; ++i) a.push_back(i*10);
// 10 20 30 40 50 60 70 80 90
it1 = it2 = a.begin();
advance (it2,6);  // ~ it2 +=6 với i vector
++it1;
it1 = a.erase(it1);  // 10 30 40 50 60 70 80 90
it2 = a.erase(it2);  // 10 30 40 50 60 80 90
++it1; --it2;
a.erase (it1,it2);  // 10 30 60 80 90
list<int> a(10,100);
```

• Xóa phần tử đầu tiên: a.pop_front();

• Xóa phần tử cuối cùng: a.pop_back();

```
int b[]= {17,89,7,14};
std::list<int> L(b,b+4);
```

• Xóa tất cả các phần tử có giá trị = 89: L.remove(89);

```
bool single_digit (const int& value) {
return (value<10); }
struct is_odd {
bool operator()(const int& value) {
```

```
return (value%2)==1; }
};
main (){
int a[]= {15,36,7,17,20,39,4,1};
list<int> L(a, a+8);  // 15 36 7 17 20 39 4 1
L.remove_if(single_digit); // 15 36 17 20 39
L.remove_if (is_odd());  // 36 20
}
list<int> a(3); // list a được khởi tạo là 3 số 0
list<int> b(5); // list b được khởi tạo là 5 số 0
b = a; // list b được gán bằng list a
a = list<int>(); // list a được gán bằng list rỗng
10. Hoán đổi giá trị giữa 2 list
list<int> a(3,300);
list<int> b(5,5000);
a.swap(b); // Hoán đổi giá trị a với b
```

Sắp xếp list

```
bool cmp(const int x, const int y){ return x > y;}
int arr[10]={3, 1, 2, 5, 4, 7, 6, 8, 10, 9);
list<int> b(arr,arr+10); a.sort(); // 1,2, .., 10
a.sort(cmp); // 10, 9, 8, ..., 1
```

Các phép toán so sánh hai list

```
int main(){
int a[]={1, 2, 3, 4}, b[]={1, 2, 3, 5};
list<int> L1(a,a+4), L2(b,b+4);
if(L1 == L2) cout<<"YES";else cout<<"NO"; //NO
if(L1 < L2) cout<<"YES";else cout<<"NO"; //YES
```

```
if(L1 <= L2) cout<<"YES";else cout<<"NO"; //YES
if(L1 > L2) cout<<"YES";else cout<<"NO"; //NO
if(L1 >= L2) cout<<"YES";else cout<<"NO"; //NO
if(L1 != L2) cout<<"YES";else cout<<"NO"; //YES
return 0;
}
```

Stack #include <stack>

(constructor)	Construct stack (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access next element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

fx Non-member function overloads

relational operators	Relational operators for stack (function)
swap (stack) C++11	Exchange contents of stacks (public member function)

fx Non-member class specializations

uses_allocator<stack> C++11	Uses allocator for stack (class template)

- Stack là loại tương thích thùng chứa được xây dựng dựa trên kiểu thùng chứa dạng dãy (vector, deque) để phục vụ các thao tác kiểu LIFO (last-in first-out), trong đó các phần tử được chèn vào và lấy ra chỉ ở một đầu của danh sách.
- Cấu trúc STL stack có khả năng chứa các phần tử thuộc các kiểu phức hợp mà C++ hỗ trợ. Các phép toán được Stack hỗ trợ gồm:
    - s.empty() : Cho biết stack có rỗng hay không.
    - s.size(): cho biết số phần tử của stack
    - s.top(): trả về phần tử trên đỉnh stack

- s.push(): chèn một phần tử vào đỉnh stack
- s.pop(): loại bỏ phần tử trên đỉnh stack

Queue #include<queue>

(constructor)	Construct queue (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
front	Access next element (public member function)
back	Access last element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove next element (public member function)
swap C++11	Swap contents (public member function)

fx Non-member function overloads

relational operators	Relational operators for queue (function)
swap (queue) C++11	Exchange contents of queues (public member function)

fx Non-member class specializations

uses_allocator<queue> C++11	Uses allocator for queue (class template)

• queue là là loại tương thích thùng chứa được xây dựng dựa trên kiểu thùng chứa dạng dãy (list, deque) được thiết kế để phục vụ các thao tác kiểu FIFO (first-in first-out), trong đó các phần tử được chèn vào ở một đầu (front) và lấy ra ở đầu kia của danh sách (back).

• Cấu trúc STL queue có khả năng chứa các phần tử thuộc các kiểu phức hợp mà C++ hỗ trợ. Các phép toán được queue hỗ trợ gồm:

- Q.empty() : Cho biết queue có rỗng hay không.
- Q.size(): cho biết số phần tử của queue
- Q.front(): trả về phần tử ở đầu của queue
- Q.back(): trả về phần tử ở cuối của queue
- Q.push(): chèn một phần tử vào cuối queue
- Q.pop(): loại bỏ phần tử đầu queue

Priority_queue #include <queue>

Member functions

(constructor)	Construct priority queue (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access top element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

Non-member function overloads

swap (queue) C++11	Exchange contents of priority queues (public member function)

• priority_queue là kiểu tương thích thùng chứa được thiết kế đặc biệt dựa trên các kiểu thùng chứa (vector, deque), để phần tử đầu tiên của nó luôn là lớn nhất, dựa trên một biểu thức thứ tự nào đó.

• priority_queue tương tự như một heap, các phần tử có thể được chèn vào ở bất kỳ thời điểm nào, và chỉ có thể lấy ra phần tử lớn nhất trong heap (là phần tử đỉnh của priority_queue)

• priority_queue có khả năng chứa các phần tử thuộc các kiểu phức hợp mà C++ hỗ trợ. Các phép toán được hỗ trợ gồm:

- PQ.empty() : Cho biết priority_queue có rỗng hay không.
- PQ.size(): cho biết số phần tử của priority_queue
- PQ.top(): trả về phần tử ở đỉnh của queue
- PQ.push(): chèn một phần tử vào cuối queue
- PQ.pop(): loại bỏ phần tử đầu queue

Ví dụ về khai báo và khởi tạo giá trị cho queue

Khai báo priority_queue chứa các số nguyên tăng dần dựa vào hàm so sánh tự viết

```
struct cmp{
bool operator() (const int x, const int y){
return x > y ;
}
```

```
};
int myints[]= {12, 55, 25, 62};
priority_queue<int, vector<int>, cmp > second(myints,myints+4);
// 12 25 55 62
```

Deque #include<queue>

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)

Modifiers:

assign	Assign container content (public member function)
push_back	Add element at the end (public member function)
push_front	Insert element at beginning (public member function)
pop_back	Delete last element (public member function)
pop_front	Delete first element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_front C++11	Construct and insert element at beginning (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)

• deque (thường được phát âm như "deck") là từ viết tắt của
double-ended queue (hàng đợi hai đầu). deque là một dạng thùng

chứa dãy với kích thước động có thể mở rộng hoặc thu hẹp ở cả hai đầu (front và back).

• Các thư viện cụ thể có thể cài đặt deque theo nhiều cách, nói chung

nó như một mảng động. Nhưng trong trường hợp nào đó, nó cho phép truy xuất trực tiếp các phần tử, và giúp khả năng thêm/bớt phần tử dễ dàng

Duyệt và truy xuất phần tử

```
for (int i=1; i < 6; i++) dq.push_back(i); // 1 2 3 4 5
for (int i = 0; i < dq.size(); i++) cout<<dq[i]<<' ';
for (int i = 0; i < dq.size(); i++) cout<<dq.at(i)<<' ';
```

deque<int>::iterator it;

for (it=dq.begin(); it!=dq.end(); ++it) cout <<' '<<*it;

dq.front() : trả về phần tử ở đầu deque

dq.back() : trả về phần tử ở cuối deque

Con trỏ ngược rbegin(), rend()

dq.clear() : xóa hết các phần tử của deque

dq.erase(it) : xóa phần tử trỏ bởi con trỏ iterator it

dq.erase(it1,it2): xóa các phần tử thuộc đoạn trỏ bởi con trỏ iterator it1 đến it2

dq.insert(it,x): chèn phần tử x vào vị trí trỏ bởi con trỏ iterator it

dq.insert(it, n, x): chèn n phần tử có giá trị x vào vị trí trỏ bởi con trỏ iterator it

dq.insert(it, it1, it2): chèn dãy phần tử [it1, it2) từ dãy khác vị trí trỏ bởi con trỏ iterator it

dq.push_back(x) : chèn phần tử x vào cuối deque

dq.push_front(x) : chèn phần tử x vào đầu deque

dq.pop_front(x) : xóa phần tử x ở đầu deque

dq.pop_back(x) : xóa phần tử x ở cuối deque

Map #include <map>

Element access:

operator[]	Access element (public member function)
at C++11	Access element (public member function)

Modifiers:

insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_hint C++11	Construct and insert element with hint (public member function)

Observers:

key_comp	Return key comparison object (public member function)
value_comp	Return value comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific key (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

• Map là loại thùng chứa kết hợp, nó cho phép lưu các phần tử dưới dạng kết hợp của một khóa và một giá trị theo một trật tự xác định.

• Trong một map, giá trị khóa thường được dung để sắp xếp và xác định duy nhất phần tử, trong khí giá trị đi kèm được lưu trong một thùng chứa kết hợp với khóa này .

• Kiểu của khóa và giá trị đi kèm có thể khác nhau, và được nhóm với nhau trong thành phần kiểu giá trị, dạng như:

typedef pair<const Key, T> value_type;

Các phần tử của map luôn được sắp xếp bởi khóa theo mộ trật tự xác định được chỉ định bởi một đối tượng hàm so sánh.

• Các giá trị được ánh xạ trong map có thể được truy cập trực tiếp bằng khóa tương ứng của chúng bằng cách sử dụng phép toán [].

• Map thường được cài đặt như cây tìm kiếm nhị phân.

Khai báo map: #include <map>

map<kiểu_khóa, kiểu_giá_trị> biến;

Dạng 1: khai báo map rỗng

```
map<char,int> mp1;
```

Dạng 3: Khai báo map với struct classcomp như một hàm so sánh

```
struct classcomp {
bool operator()(const char& c1, const char& c2) const
{return c1 < c2;}
};
map <char, int, classcomp> mp4;
```

Chèn phần tử vào map

Ví dụ 1: chèn giá trị bằng phép gán

```
map<char,int> mp1;
mp1['a']=70; mp1['b']=30; mp1['c']= 50; mp1['d']= 10;
map<char,int>::iterater it;
for(it = mp1.begin(); it!=mp1.end(); it++)
cout<<' ' << it1->first << ' ' << it->second;
//a 70 b 30 c 50 d 10
```

Ví dụ 2: chèn giá trị bằng hàm insert(), make_pair()

```
map<char,int> mp;
mp.insert (pair<char,int>('a',200) );
mp.insert (make_pair('z',100));
pair<map<char,int>::iterator,bool> ret;
ret = mp.insert (pair<char,int>('z',100) );
if (ret.second == false)
cout << "'z' da co gia tri "<< ret.first->second;
```

Ví dụ 3: chèn giá trị qua vị trí con trỏ iterater

```
map<char,int> mp;
mp.insert (make_pair('z',100));
```

```
map<char,int>::iterator it = mp.begin();
mp.insert (it, pair<char,int>('b',300));
mp.insert (it, pair<char,int>('c',400));
//b 300 c 400 z 100
```

Các giá trị luôn được sắp xếp

các hàm về kích thước, xóa phần tử và tráo đổi map

mp.clear(): xóa tất cả các phần tử trong map O(n)

mp.empty(): kiểm tra map rỗng O(1)

mp.size(): trả về kích thước map O(1)

mp.erase(key): xóa phần tử trong map có khóa là key

mp.erase(it): xóa phần tử trong map được trỏ bởi it

mp.erase(it1, it2): xóa tất cả các phần tử trong map thuộc đoạn [it1, it2)

mp.swap(mp1): hoán đổi phần tử hai map O(n)

Hai hàm erase() đầu có O(log2n), hàm sau có O(n)

các hàm đếm, tìm kiếm phần tử map

mp.count(key): đếm số phần tử có khóa bằng key it = mp.find(key): trả về iterater của phần tử có khóa bằng key (nếu không thấy thì trả về mp.end())

Các hàm này đều có O(log2n)

hàm tìm kiếm phần tử map equal_range() - O(log2n)

```
map<char,int> mp;
mp['a']=10; mp['b']=20; mp['c']=30;
typedef map<char,int>::iterator mpit;
pair<mpit, mpit> ret;
ret = mp.equal_range('b');
cout << "lower bound : ";
cout << ret.first->first << " " << ret.first->second <<' ';
cout << "upper bound : ";
```

```
cout << ret.second->first <<" "<< ret.second->second;

//lower bound : b 20 upper bound : c 30
```

hàm tìm kiếm phần tử lower_bound(), upper_bound()

mp.lower_bound(key): Trả về iterater của phần tử có khóa không nhỏ hơn key (nếu không có thì trả về mp.end()).

mp.upper_bound(key): Trả về iterater của phần tử có khóa lớn hơn key (nếu không có thì trả về mp.end()).

hàm tìm kiếm phần tử lower_bound(), upper_bound()

Ví dụ:

```
map<char,int> mp;

map<char,int>::iterator low, up, it;

mp['a']=20; mp['b']=40;

mp['c']=60; mp['d']=80; mp['e']=100;

low=mp.lower_bound('b'); up=mp.upper_bound('c');

mp.erase(itlow,itup);

for (it=mp.begin(); it!=mp.end(); ++it)

cout << it->first << "=>" << it->second << ', ';

// a=>20, d=>80, e=>100
```

Set #include <set>

Modifiers:

insert	Insert element (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_hint C++11	Construct and insert element with hint (public member function)

Observers:

key_comp	Return comparison object (public member function)
value_comp	Return comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific value (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

• Set là loại thùng chứa được thiết kế để lưu các phần tử đơn trị theo một trật tự xác định.

• Trong một set, giá trị của phần tử được nhận diện bởi chính nó (giá trị của nó chính là khóa để tìm nó), mỗi giá trị phải là duy nhất. Giá trị của các phần tử trong set không thể bị thay đổi (luôn là hằng), nhưng chúng có thể được chèn vào hay loại bỏ khỏi set.

• Phần tử của set luôn được sắp xếp theo một trật tự thông qua đối tượng hàm so sánh của nó.

• Mặc định trong set trật tự luôn là tăng dần. Ta có thể duyệt các phần tử của set thông qua con trỏ iterater của nó.

• Set thường được cài đặt như cây tìm kiếm nhị phân

Khai báo map: #include <set>

set <kiểu_giá_trị> biến;

Dạng 2: khai báo set với đối tượng hàm so sánh trật tự

struct comp {

bool operator() (const int& x, const int& y) const

{return x < y;}

};

```
set<int, comp> s5;
```

Chèn phần tử vào set

```
set<int> s; set<int>::iterator it;
pair<set<int>::iterator,bool> ret;
for (int i=1; i<=5; ++i) s.insert(i*10);
// set: 10 20 30 40 50
ret = myset.insert(20);
if (ret.second==false) it=ret.first;
// vị trí "it" đã có giá trị 20
s.insert (it,25);
s.insert (it,24);
s.insert (it,26);
int a[]= {5,10,15}; s.insert (a,a+3);
// số 10 đã có trong set và ko được chèn vào
for (it = s.begin(); it!= s.end(); ++it)
cout<<' '<<*it;
// 5 10 15 20 24 25 26 30 40 50
```

các hàm về kích thước, xóa phần tử và tráo đổi set

```
s.clear(): xóa tất cả các phần tử trong set O(n)
s.empty(): kiểm tra set rỗng O(1)
s.size(): trả về kích thước set O(1)
s.erase(key): xóa phần tử trong set có khóa là key
s.erase(it): xóa phần tử trong set được trỏ bởi it
s.erase(it1, it2): xóa tất cả các phần tử trong set
thuộc đoạn [it1, it2)
s.swap(s1): hoán đổi phần tử hai set O(n)
Hai hàm erase() đầu có O(log2n), hàm sau có O(n)
```

các hàm đếm, tìm kiếm phần tử set

s.count(key): đếm số phần tử có khóa bằng key

it = s.find(key): trả về iterater của phần tử có khóa

bằng key (nếu không thấy thì trả về mp.end())

Các hàm này đều có O(log2n)

hàm tìm kiếm phần tử set equal_range() - O(log2n)

```
set<int> s;

for (int i=1; i<=5; i++) s.insert(i*10); //s: 10 20 30 40 50

typedef set<int>::const_iterator sit;

pair<sit, sit> ret;

ret = s.equal_range(30);

cout << "lower bound: " << *ret.first << '\n';

cout << "upper bound : " << *ret.second << '\n';

//lower bound : 30 upper bound : 40
```

hàm tìm kiếm phần tử lower_bound(), upper_bound()

s.lower_bound(key): Trả về iterater của phần tử có khóa không nhỏ hơn key (nếu không có thì trả về s.end()).

s.upper_bound(key): Trả về iterater của phần tử có khóa lớn hơn key (nếu không có thì trả về s.end()).

hàm tìm kiếm phần tử lower_bound(), upper_bound()

```
set<int> s;

set<int>::iterator low, up, it;

for (int i=1; i<10; i++) myset.insert(i*10);

// 10 20 30 40 50 60 70 80 90

low = myset.lower_bound(30);

up=myset.upper_bound (60);

// 10 20 30 40 50 60 70 80 90

myset.erase(low,up); // 10 20 70 80 90

for(it=myset.begin(); it!=myset.end(); ++it)
```

```
cout << ' ' << *it;
```