

# C++ STL 库简单入门

不熟悉 c++ 标准库，任何人都称不上是高生产力的 c++ 程序员

# 目录

Stack.....	2
定义一个栈: .....	3
1.1 操作.....	3
push:向栈中压入新的元素.....	3
top: 返回栈顶元素的值 .....	3
pop:将栈顶元素弹出 .....	4
size:返回栈中元素的个数.....	5
empty:判断栈是否为空.....	5
Queue.....	5
定义一个队列: .....	5
操作.....	6
优先队列——priority_queue.....	6
Deque.....	7
定义一个双向队列 .....	8
操作: .....	8
插入: .....	8
删除: .....	8
查找: .....	8
iterator: .....	9
Vector .....	9
定义一个 vector .....	9
操作: .....	9
非更易型操作 .....	9
赋值和访问.....	10
迭代器相关: .....	11
安插和删除.....	11
List.....	11
定义一个 list.....	11
操作: .....	12
非更易型操作 .....	12
访问 .....	12
迭代器相关.....	12
安插和移除.....	12
Set .....	13
定义一个 set .....	13
操作.....	13
非更易型操作 .....	13
迭代器相关.....	14
查找 .....	14

# C++ STL 库简单入门

## C++ STL Simple introduction

### 摘要:

“不熟悉 c++ 标准库，任何人都称不上是高生产力的 c++ 程序员”

——摘自 c++ 标准库（第二版）

作为一个计算机系的大学生，个人认为有必要学好一门比较流行的编程语言，相信不少同学都选择了 c++ 这门编程语言。而个人认为想要学好 c++，c++ 标准库即 c++ STL 库是有必要了解学习一下的。

C++ STL 库里面存放了好多已经实现好了功能的 container（容器），algorithm（算法），iterator（迭代器），adapter（适配器），function object（函数对象），allocator（分配器）。然而我们并不打算讲这么多，我们只是打算讲一下一些最简单的，最常用的容器。

作为一名计算机的学生估计都学过一些 c++ 的基础东西，例如：class（类），template（模版）。这些东西就我们要讲的东西的基础，因为标准库本身的实现就用到了这些东西。

### Abstract:

#### Standard Containers

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows a great flexibility in the types supported as elements.

The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers).

关键词: C++, STL 库

## Stack

对于我们计算机系的学生来说，估计都对栈这个数据结构不陌生，然而在 C++ STL 库中已经定义好了这么一种容器——stack。

首先我们必须在头文件里包含 <stack> 这么一个头文件。即

```
#include <stack>
```

我们所有需要的有关这个容器的操作都存放在了这个头文件里面了。

## 定义一个栈:

如果我们要定义一个 int 型的栈:

```
stack<int> s;
```

int 是指的新定义的栈的数据类型, s 指的新定义的栈的名称。之前已经说过了 STL 库中的容器基本都是用模板和类实现的, 所以我们完全可以任意定义栈的数据类型(当然要保证同一个栈中的数据类型必须相同), 甚至可以是结构体或类。

### 1.1 操作

#### push: 向栈中压入新的元素

因为所有新定义的 stack 中都没有新的元素, 我们要想新定义的栈中压入新元素就要用到 push.

```
1  #include<iostream>
2  #include<stack>
3
4  using namespace std;
5
6  int main(){
7      stack<int>s;           //新定义一个int型的栈
8      s.push(1);           //向栈中压入一个整数 1
9
10
11     return 0;
12 }
```

D:\源代码\acm\未命名1.exe

```
-----
Process exited after 0.4834 seconds with return value 0
请按任意键继续. . .
```

此时我们只是压入了一个整数 1, 并未做任何打印, 所以运行结果什么都没有。

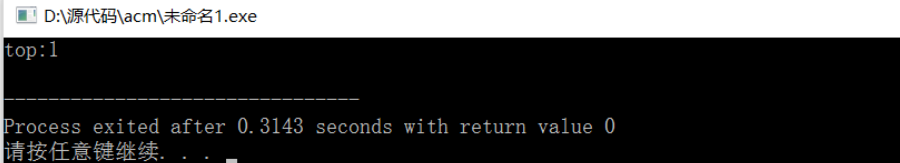
#### top: 返回栈顶元素的值

当我们使用 push 的时候并未返回任何打印结果, 我们在上面的代码中加入这么一个语句

```
cout << s.top();
```

就可以把原先压入栈中的元素输出了:

```
1 #include<iostream>
2 #include<stack>
3
4 using namespace std;
5
6 int main(){
7     stack<int>s;    //新定义一个int型的栈
8     s.push(1);    //向栈中压入一个整数 1
9     cout << "top:" << s.top() << endl;    //输出栈顶元素的值
10
11     return 0;
12 }
```

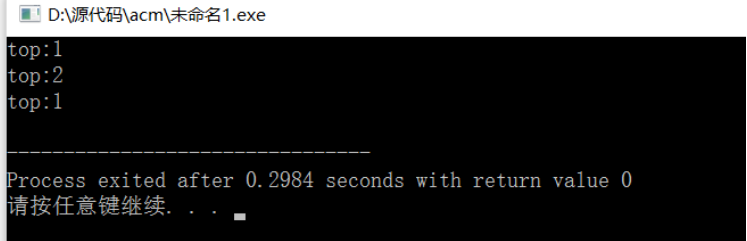


应当注意的是：由于栈这个数据结构的特殊性，我们只能返回栈顶元素的值而无法访问其他元素的值。

## pop:将栈顶元素弹出

同 push 相反，pop 是将栈顶元素移除：

```
1 #include<iostream>
2 #include<stack>
3
4 using namespace std;
5
6 int main(){
7     stack<int>s;    //新定义一个int型的栈
8     s.push(1);    //向栈中压入一个整数 1
9     cout << "top:" << s.top() << endl;    //输出栈顶元素的值
10    s.push(2);
11    cout << "top:" << s.top() << endl;
12    s.pop();    //移除新压入的元素 2
13    cout << "top:" << s.top() << endl;
14
15    return 0;
16 }
```



## size: 返回栈中元素的个数

```
1 #include<iostream>
2 #include<stack>
3
4 using namespace std;
5
6 int main(){
7     stack<int>s;    //新定义一个int型的栈
8     cout << "size:" << s.size() << endl;
9     cout << endl;
10    s.push(1); //向栈中压入一个整数 1
11    cout << "size:" << s.size() << endl;
12    cout << "top:" << s.top() << endl; //输出栈顶元素的值
13    cout << endl;
14    s.push(2);
15    cout << "size:" << s.size() << endl;
16    cout << "top:" << s.top() << endl;
17    cout << endl;
18    s.pop(); //移除新压入的元素 2
19    cout << "size:" << s.size() << endl;
20    cout << "top:" << s.top() << endl;
21
22    return 0;
23 }
```

```
D:\源代码\acm\未命名1.exe
size:0
size:1
top:1
size:2
top:2
size:1
top:1
-----
Process exited after 0.2185 seconds with return value 0
请按任意键继续. . .
```

就像图中所示，size 可以返回当前栈中元素的数量

## empty: 判断栈是否为空

empty 可以用来判断栈是否为空如果为空则返回 true，否则则返回 false。

```
1 #include<iostream>
2 #include<stack>
3
4 using namespace std;
5
6 int main(){
7     stack<int>s;    //新定义一个int型的栈
8     cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
9     cout << endl;
10    s.push(1); //向栈中压入一个整数 1
11    cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
12    cout << "top:" << s.top() << endl; //输出栈顶元素的值
13    cout << endl;
14    s.push(2);
15    cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
16    cout << "top:" << s.top() << endl;
17    cout << endl;
18    s.pop(); //移除新压入的元素 2
19    cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
20    cout << "top:" << s.top() << endl;
21    cout << endl;
22    s.pop();
23    cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
24    return 0;
25 }
```

```
D:\源代码\acm\未命名1.exe
size:0 empty:1
size:1 empty:0
top:1
size:2 empty:0
top:2
size:1 empty:0
top:1
size:0 empty:1
-----
Process exited after 0.2926 seconds with return value 0
请按任意键继续. . .
中文 - QQ拼音输入法 半
```

## Queue

与栈的后进出不同，队列是后进前出，可以读取队列最前端的元素的数值和队列尾端的元素数值。需要的头文件：<queue>。

### 定义一个队列：

同栈一样定义一个队列需要指定数据类型，例如定义一个 int 型队列的队列：

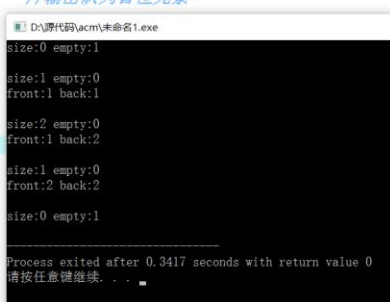
```
queue<int> qu;
```

## 操作

同 stack 相同，queue 中也有着 push, pop, size, empty 的操作。同时 queue 中也多了 back 和 front 操作。

1. push: 在队列尾部加入一个新的元素
2. pop: 从队列首端弹出一个元素
3. size: 返回队列中元素的个数
4. empty: 检查队列是否为空，若为空返回 true，否则返回 false
5. back: 返回队列尾端的元素
6. front: 返回队列首端的元素

```
1 #include<iostream>
2 #include<queue>
3
4 using namespace std;
5
6 int main(){
7     queue<int> s; //新定义一个int型的队列
8     cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
9     cout << endl;
10    s.push(1); //向队列中加入一个整数 1
11    cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
12    cout << "front:" << s.front() << " " << "back:" << s.back() << endl; //输出队列首位元素
13    cout << endl;
14    s.push(2);
15    cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
16    cout << "front:" << s.front() << " " << "back:" << s.back() << endl;
17    cout << endl;
18    s.pop(); //移除队列首端元素
19    cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
20    cout << "front:" << s.front() << " " << "back:" << s.back() << endl;
21    cout << endl;
22    s.pop();
23    cout << "size:" << s.size() << " " << "empty:" << s.empty() << endl;
24    return 0;
25 }
```



同 stack 一样，queue 操作起来也很方便。

## 优先队列——priority\_queue

然而在<queue>中还定义着这么一种数据容器，优先队列

- 1 C++优先队列类似队列，但是在这个数据结构中的元素按照一定的断言排列有序。

1. empty 如果优先队列为空，则返回真
2. pop 删除第一个元素
3. push 加入一个元素
4. size 返回优先队列中拥有的元素的个数
5. top 返回优先队列中有最高优先级的元素

优先队列默认为按照降序排列

```

1 #include <iostream>
2 #include <queue>
3 using namespace std;
4
5 class T {
6 public:
7     int x, y, z;
8     T(int a, int b, int c):x(a), y(b), z(c)
9     {
10     }
11 };
12 bool operator < (const T &t1, const T &t2)
13 {
14     return t1.z < t2.z; // 按照z的顺序来决定t1和t2的顺序
15 }
16 int main()
17 {
18     priority_queue<T> q;
19     q.push(T(4,4,3));
20     q.push(T(2,2,5));
21     q.push(T(1,5,4));
22     q.push(T(3,3,6));
23     while (!q.empty())
24     {
25         T t = q.top();
26         q.pop();
27         cout << t.x << " " << t.y << " " << t.z << endl;
28     }
29     return 0;
30 }

```

```

D:\源代码\acm\未命名1.exe
3 3 6
2 2 5
1 5 4
4 4 3

Process exited after 0.2341 seconds with return value 0
请按任意键继续. . .
中文 - QQ拼音输入法 半 :

```

如果想要按照升序排列需要加个 greater 函数

```

1 #include <iostream>
2 #include <queue>
3 using namespace std;
4 class T {
5 public:
6     int x, y, z;
7     T(int a, int b, int c):x(a), y(b), z(c) {}
8 };
9 bool operator > (const T &t1, const T &t2) {
10     return t1.z > t2.z;
11 }
12 int main() {
13     priority_queue<T, vector<T>, greater<T> > q;
14     q.push(T(4,4,3));
15     q.push(T(2,2,5));
16     q.push(T(1,5,4));
17     q.push(T(3,3,6));
18     while (!q.empty())
19     {
20         T t = q.top();
21         q.pop();
22         cout << t.x << " " << t.y << " " << t.z << endl;
23     }
24     return 0;
25 }

```

```

D:\源代码\acm\未命名1.exe
4 4 3
1 5 4
2 2 5
3 3 6

Process exited after 0.2385 seconds with return value 0
请按任意键继续. . .

```

## Deque

相比于 queue，deque 可以从前端插入元素，从后端弹出元素。

要求包含头文件：<deque>

deque 的特点：

- (1) 随机访问方便，即支持[] 操作符和 vector.at() ，但性能没有 vector 好；
- (2) 可以在内部进行插入和删除操作，但性能不及 list ；
- (3) 可以在两端进行 push 、 pop ；
- (4) 相对于 verctor 占用更多的内存。

双向队列和向量很相似，但是它允许在容器头部快速插入和删除（就像在尾部一样）。



## 定义一个双向队列

<code>deque&lt;T&gt;a</code>	定义一个新的空 deque
<code>deque&lt;T&gt;a(n)</code>	定义一个大小为 n 的 deque
<code>a.~deque()</code>	销毁所有元素，释放内存

## 操作：

同之前一样，含有返回元素数量的 `size` 函数和判断是否为空的 `empty` 函数。  
可以用 `clear` 来清空整个双向队列。

## 插入：

1. `push_back` 从后端插入
2. `push_front` 从前端插入

因为 deque 可以从前后两端插入，所以在 deque 中有两个插入函数。

## 删除：

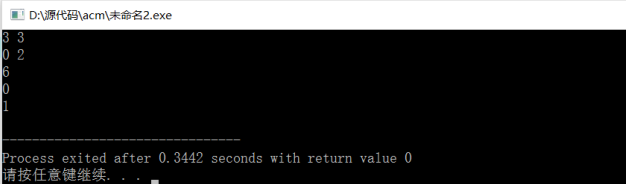
同插入一样有两个删除函数：

1. 删除前端元素 `pop_front`;
2. 删除后端元素 `pop_back`;

## 查找：

同 queue 一样，可以用 `front` 和 `back` 函数实现对前端元素和后端元素的查找。


```
1 #include<iostream>
2 #include<deque>
3 using namespace std;
4 int main(){
5     deque<int> s;
6     for(int i = 1; i < 4; i++){ //分别从前后两端插入1 2 3, 执行之后的队列中的元素为 3,2,1,1,2,3;
7         s.push_back(i);
8         s.push_front(i);
9     }
10    cout << s.front() << " " << s.back() << endl; //查看当前的首尾元素
11    s.pop_back(); s.push_front(0); //删除当前尾端元素, 从前端插入0, 此时队列中元素为 0,3,2,1,1,2;
12    cout << s.front() << " " << s.back() << endl;
13    cout << s.size() << endl; //输出当前队列中元素的个数
14    s.clear(); //清空队列
15    cout << s.size() << endl;
16    cout << s.empty() << endl;
17    return 0;
18 }
```



## iterator:

1. begin 返回开始位置的 iterator。
2. end 返回结束位置的 iterator。（返回的是最后一个元素的下一位置）

```
1 #include<iostream>
2 #include<deque>
3 using namespace std;
4 int main(){
5     deque<int> s;
6     for(int i = 1; i < 4; i++){ //分别从前后两端插入1 2 3, 执行之后的队列中的元素为 3,2,1,1,2,3;
7         s.push_back(i);
8         s.push_front(i);
9     }
10    deque<int>::iterator it; //定义一个迭代器
11    for(it = s.begin(); it < s.end(); it++){ //使用迭代器输出当前队列中的所有元素
12        cout << *it << " ";
13    }
14    cout << endl;
15    cout << s.front() << " " << s.back() << endl; //查看当前的首尾元素
16    s.pop_back(); s.push_front(0); //删除当前尾端元素, 从前端插入0, 此时队列中元素为 0,3,2,1,1,2;
17    cout << s.front() << " " << s.back() << endl;
18    cout << s.size() << endl; //输出当前队列中元素的个数
19    s.clear(); //清空队列
20    cout << s.size() << endl;
21    cout << s.empty() << endl;
22    return 0;
23 }
```



## Vector

需要头文件: <vector>

vector 容器是一个模板类, 可以存放任何类型的对象 (但必须是同一类对象)。vector 对象可以在运行时高效地添加元素, 并且 vector 中元素是连续存储的。

### 定义一个 vector

vector<T> c	建立一个新的空 vector
vector<T>c (n)	建立一个大小为 n 的 vector
c.~vector()	销毁所有元素, 释放内存

### 操作:

### 非更易型操作

1. c.empty (): 判断容器是否为空;
2. c.size (): 返回当前容器内元素的个数;
3. c.max\_size (): 返回元素个数之最大可能量;
4. c.capacity (): 返回“不进行空间重新分配”条件下的最大容纳量;
5. c.reserve (num): 如果容量不足, 扩大之;

```

1 #include<iostream>
2 #include<vector>
3 using namespace std;
4 int main(){
5     vector<int> coll(100); //定义一个大小为100的vector
6     cout << coll.size() << endl; //输出当前vector的大小
7     cout << coll.max_size() << endl; //输出当前vector的最大可能量
8     cout << coll.capacity() << endl; //输出当前vector的最大容纳量
9     coll.reserve(150); //扩大至为能容纳150的vector
10    cout << coll.size() << endl;
11    cout << coll.max_size() << endl;
12    cout << coll.capacity() << endl;
13    return 0;
14 }

```

```

D:\源代码\acm\未命名3.exe
100
1073741823
100
100
1073741823
150

-----
Process exited after 0.2254 seconds with return value 0
请按任意键继续. . .

```

## 赋值和访问

1. `c = c2` : 将 `c2` 中的元素赋值给 `c`
2. `c.assign(n, elem)`: 复制 `n` 个 `elem`, 赋值给 `c`
3. `c.assign(beg, end)`: 将区间 `[beg, end)` 内的元素赋值给 `c`
4. `c1.swap(c2)` 等同 `swap(c1, c2)`: 置换 `c1, c2` 的元素
5. `c[idx]`: 返回 `idx` 所指的元素 (不检查是否超出范围)
6. `c.at(idx)`: 返回 `idx` 所指的元素 (若超出范围就返回异常)
7. `c.front()`: 返回第一个元素
8. `c.back()`: 返回最后一个元素

```

1 #include<iostream>
2 #include<vector>
3 using namespace std;
4 int main(){
5     vector<int> c(10), c2(10); //定义一个大小为10的vector
6     for(int i = 0; i < 10; i++){ //为 c赋值并输出
7         c[i] = i;
8         cout << c[i] << " ";
9     } cout << endl;
10    c2.assign(10, 5); //将c2中的10个元素赋值为5
11    for(int i = 0; i < 10; i++) cout << c2.at(i) << " "; cout << endl;
12    c.swap(c2); //交换c和c2
13    for(int i = 0; i < 10; i++) cout << c.at(i) << " "; cout << endl;
14    for(int i = 0; i < 10; i++) cout << c2.at(i) << " "; cout << endl;
15    c.assign(c2.begin() + 5, c2.end()); //将c2中的6-10位元素赋给c
16    for(int i = 0; i < 5; i++) cout << c.at(i) << " "; cout << endl;
17    cout << c.front() << " " << c.back() << endl;
18    return 0;
19 }

```

```

D:\源代码\acm\未命名3.exe
0 1 2 3 4 5 6 7 8 9
5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5
0 1 2 3 4 5 6 7 8 9
5 6 7 8 9
5 9

-----
Process exited after 0.3172 seconds with return value 0
请按任意键继续. . .

```

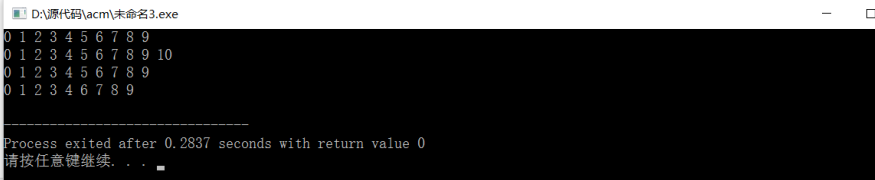
## 迭代器相关:

1. `c.begin()` 返回指向第一个元素的 `iterator`
2. `c.end()` 返回指向第二个元素的 `iterator`

## 安插和删除

1. `c.push_back(x)`: 在末尾插入一个 `x` 元素
2. `c.pop_back(x)`: 移除末尾元素但不返回值
3. `c.insert(pos, x)`: 在 `iterator` 之前位置插入一个 `x` 元素
4. `c.erase(pos)`: 移除 `iterator` 位置 `pos` 上的元素
5. `c.resize(num)`: 将元素数量改为 `num`
6. `c.clear()`: 将容器清空

```
1 #include<iostream>
2 #include<vector>
3 using namespace std;
4 int main(){
5     vector<int> c(10); //定义一个大小为10的vector
6     for(int i = 0; i < 10; i++){
7         c[i] = i;
8         cout << c[i] << " ";
9     } cout << endl;
10    c.push_back(10); //从尾端插入 10
11    for(vector<int>::iterator it = c.begin(); it < c.end(); it++) cout << *it << " "; cout << endl;
12    c.pop_back(); //删除尾端元素
13    for(vector<int>::iterator it = c.begin(); it < c.end(); it++) cout << *it << " "; cout << endl;
14    c.erase(c.begin() + 5); //删除第6个元素
15    for(vector<int>::iterator it = c.begin(); it < c.end(); it++) cout << *it << " "; cout << endl;
16    return 0;
17 }
```



## List

`List` 是一个双向循环链表, 所以每一个 `list` 中的元素都指向其前一个元素和其后一个元素。

1. `list` 不支持随机访问, 在 `list` 中随机访问某个元素是个很缓慢的行为。
2. 任何位置上安插或移除元素都很快, 时间复杂度是  $O(1)$
3. 安插或者删除元素不会造成各个 `pointer`, `iterator` 失效

## 定义一个 `list`

<code>list&lt;T&gt; c</code>	创建一个空的 <code>list</code> , 没有任何元素
<code>list&lt;T&gt; c(n)</code>	创建一个大小为 <code>n</code> 的 <code>list</code>
<code>c.~list</code>	销毁所有元素, 释放内存

## 操作:

## 非更易型操作

1. `c.empty()`: 判断容器是否为空
2. `c.size()`: 返回目前元素的个数
3. `c.max_size()`: 返回 list 能容纳的最大元素数量

## 访问

1. `c.front()`: 返回第一个元素
2. `c.back()`: 返回第二个元素

以上操作都不检查容器是否为空, 对空容器执行操作都会导致不明行为, 所以调用者必须保证容器中至少存在一个元素

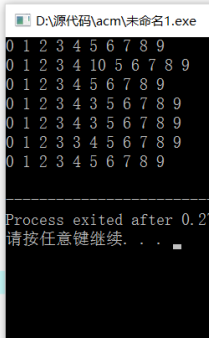
## 迭代器相关

- `c.begin()`: 返回指向第一个元素的迭代器  
`c.end()`: 返回指向最后一个元素下一个位置的迭代器

## 安插和移除

1. `c.push_back(x)`: 在容器末尾插入一个 x 元素
2. `c.pop_back()`: 移除最后一个元素
3. `c.push_front(x)`: 在容器头部插入一个 x 元素
4. `c.pop_front(x)`: 移除第一个元素
5. `c.insert(pos, x)`: 在 pos 所指位置前插入 x 元素并返回新元素所在位置
6. `c.erase(pos)`: 移除 pos 所指位置的元素并返回指向下一元素的 iterator
7. `c.clear()`: 清空链表
8. `c.unique()`: 消除重复元素
9. `c.sort()`: 以<为准则排序对所有元素排序
10. `c.sort(op)`: 以 op() 为准则对所有元素进行排序
11. `c.merge(c2)`: 合并 c 和 c2 (如果 c 和 c2 都是排序完的, 新合并的 list 也是合并好的)

```
1 #include<iostream>
2 #include<list>
3 using namespace std;
4 void print(list<int> &a){ //输出当前list容器中的所有元素
5     for(list<int>::iterator it = a.begin(); it != a.end(); it++) cout << *it << " ";
6     cout << endl;
7 }
8 int main(){
9     list<int> c;
10    for(int i = 0; i < 10; i++) c.push_back(i); //为c赋值
11    print(c); //输出
12    list<int>::iterator it = c.begin(); for(int i = 0; i < 5; i++) it++;
13    it = c.insert(it, 10); print(c); //在第6个位置加入一个元素10
14    it = c.erase(it); print(c); //删除刚刚加入的元素
15    c.insert(it, 3); print(c);
16    c.unique(); print(c); //重复数字不连续的时候unique不生效
17    c.sort(); print(c); //排序;
18    c.unique(); print(c); //排序后unique函数生效
19    return 0;
20 }
```



## Set

头文件: <set>

实现了红黑树的平衡二叉检索树的数据结构,插入元素时,它会自动调整二叉树的排列,把元素放到适当的位置,以保证每个子树根节点键值大于左子树所有节点的键值,小于右子树所有节点的键值;另外,还得保证根节点左子树的高度与右子树高度相等。

Set 会根据特定的排序准则,自动将元素排序,并且 set 不允许元素重复。

所谓的排序准则:

1. 必须是非对称的,若  $x < y == \text{true}$ , 则  $x > y == \text{false}$ 。
2. 必须是可传递的,若  $x < y == \text{true} \ \&\& \ y < z == \text{true}$ , 则  $x < z == \text{true}$ 。
3. 必须是非自反的,即  $x < x == \text{false}$

## 定义一个 set

`set<T> c` : 定义一个 set, 默认为以 (operator) 为准则排序

`set<T, op> c`: 定义一个 set, 以 `op()` 为准则排序

## 操作

### 非更易型操作

1. `c.empty()`: 判断容器是否为空
2. `c.size()`: 返回目前容器内元素的个数

## 迭代器相关

1. `c.begin()`: 返回指向第一个元素的 iterator
2. `c.end()`: 返回指向最后一个元素下一位置的 iterator
3. `c.insert(x)`: 向 set 容器中插入 x
4. `c.erase(x)`: 移除 set 中的 x 元素
5. `c.clear()`: 清空容器

## 查找

`c.find(val)`: 返回“元素之为 val”的第一个元素，如果找不到就返回 `end()`

```
1  #include<iostream>
2  #include<set>
3  using namespace std;
4  void print(set<int> &a){    //输出当前set容器中储存的元素
5      for(set<int>::iterator it = a.begin(); it != a.end(); it++)
6          cout << *it << " "; cout << endl;
7  }
8  int main(){
9      set<int> c; int a[10] = {5, 4, 1, 3, 2, 6, 9, 0, 8, 7};
10     for(int i = 0; i < 10; i++) c.insert(a[i]); print(c);
11     //将数组a中的元素插入到容器c中 (明显是已经排好序的数组)
12     c.erase(5); print(c);    //删除容器中的元素5
13     cout << *c.find(4) << endl; //find函数返回的是元素4 的位置
14     return 0;
15 }
```

选择D:\源代码\acm\未命名1.exe

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 6 7 8 9
4
```

Process exited after 0.3225 seconds with return value 0  
请按任意键继续. . .