<https://blog.csdn.net/xiaoquantouer/article/details/70339869>

头文件

#include<list>

声明一个int型的list：list<int> a；

**1、list的构造函数**

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>a{1,2,3}
2. list<**int**>a(n)    //声明一个n个元素的列表，每个元素都是0
3. list<**int**>a(n, m)  //声明一个n个元素的列表，每个元素都是m
4. list<**int**>a(first, last)  //声明一个列表，其元素的初始值来源于由区间所指定的序列中的元素，first和last是迭代器

**2、begin()和end()**

通过调用list容器的成员函数begin()得到一个指向容器起始位置的iterator，可以调用list容器的end()函数来得到list末端下一位置

**3、push\_back()和push\_front()**

使用list的成员函数push\_back和push\_front插入一个元素到list中。其中push\_back()是从list的末端插入，而push\_front()是从list的头部插入。

**4、empty()**

判断list是否为空

**5、resize()**

调用resize(n)将list的长度改为只容纳n个元素，超出的元素将被删除。如果n比list原来的长度长，那么默认超出的部分元素置为0。也可以用resize(n, m)的方式将超出的部分赋值为m。

例子：

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>b{1, 2, 3, 4};
2. b.resize(2);
3. list中输出元素：1,2
4. list<**int**>b{1, 2, 3, 4};
5. b.resize(6);
6. list中输出元素：1,2,3,4,0,0
7. list<**int**>b{1, 2, 3, 4};
8. b.resize(6,9);
9. list中输出元素：1,2,3,4,9,9

**6、clear()**

清空list中的所有元素

**7、front()和back()**

通过front()可以获得list容器中的头部元素，通过back()可以获得list容器的最后一个元素。注意：当list元素为空时，这时候调用front()和back()不会报错。因此在编写程序时，最好先调用empty()函数判断list是否为空，再调用front()和back()函数。

**8、pop\_back()和pop\_front()**

使用pop\_back()可以删掉尾部第一个元素，pop\_front()可以删掉头部第一个元素。注意：list必须不为空，如果当list为空的时候调用pop\_back()和pop\_front()会使程序崩掉。

**9、assign()**

有两种使用情况：

（1）a.assign(n, val):将a中的所有元素替换成n个val元素

例如：

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>b{1,2,3,4,5};
2. b.assign(5,10);

b中的元素变为10, 10, 10, 10, 10

（2）a.assign(b.begin(), b.end())

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>a{6,7,8,9};
2. list<**int**>b{1,2,3,4,5};
3. b.assign(a.begin(),a.end());

b中的元素变为6,7,8,9

**10、swap()**

交换两个链表。a.swap(b)和swap(a, b)，都可以完成a链表和b链表的交换。

例子：

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>a{6,7,8,9};
2. list<**int**>b{1,2,3,4,5};
3. swap(a, b);  //或a.swap(b)

a中元素变为1,2,3,4,5

b中元素变为6,7,8,9

**11、reverse()**

可以实现list的逆置

例子：

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>b{1,2,3,4,5};
2. reverse(b.begin(),b.end());

b中元素变为5，4，3，2，1

**12、merge()**

a.merge(b) 调用结束后b变为空，a中元素包含原来a和b的元素。

例子：

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>a{6,7,8,9};
2. list<**int**>b{2, 1, 3, 6, 5};
3. a.merge(b,greater<**int**>());

a中元素变为：6,7,8,9,2,1,3,6,5

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>a{6,7,8,9};
2. list<**int**>b{2, 1, 3, 6, 5};
3. a.merge(b);

a中元素变为：2,1,3,6,5,6,7,8,9

**13、insert()**

在指定位置插入一个或多个元素

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. a.insert(a.begin(),100);  //在a的开始位置（即头部）插入100
2. a.insert(a.begin(),2, 100);   //在a的开始位置插入2个100
3. a.insert(a.begin(),b.begin(), b.end());//在a的开始位置插入b从开始到结束的所有位置的元素

**14、erase()**

删除一个元素或一个区域的元素

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. a.erase(a.begin());  //将a的第一个元素删除
2. a.erase(a.begin(),a.end());  //将a的从begin()到end()之间的元素删除。

**15、remove()函数**

从list中删除元素

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. list<**int**>a{6,7,8,9,7,10};
2. a.remove(7);

删除了a中所有值为7的元素，此时a中元素为6,8,9,10

**16、remove\_if()函数**

括号中可以传入

（1）回调函数

回调函数的原型为boolisRemove(T &obj1);

函数名任意，如果obj1需要被移除则返回1，否则返回0

使用方法：list.remove\_if(isRemove)

这种方法最简单，但是无法向回调函数中传递参数，每一个条件就要有一个回调函数，因此不推荐使用

（2）创建用于比较的类，传入类名及初始化参数

用于比较的类必须重载bool operator()(T &obj1)方法，如果obj1需要被移除则返回1，否则返回0.

用于比较的类还应当包含必要的构造函数，用于传递参数。

使用方法：list.remove\_if(classname(args))

例1：

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. **bool** is\_odd(constint& value){
2. **return** (value==4);
3. }
5. **int** main(){
6. list<**int**> a{6,7,4,9,7,10};
7. a.remove\_if(is\_odd);
9. list<**int**>::iterator it = a.begin();
10. **while**(it != a.end()){
11. cout<<\*it<< " ";
12. it++;
13. }

16. **return** 0;
17. }

输出：

6 7 9 7 10

例2：

**[cpp]** [view plain](https://blog.csdn.net/xiaoquantouer/article/details/70339869) [copy](https://blog.csdn.net/xiaoquantouer/article/details/70339869)

1. **class** single\_digit{
2. **public**:
3. **bool** operator()(**const** **int**& value){
4. **return** (value<10);
5. }
6. };
8. **int** main(){
9. list<**int**> a{6,7,4,9,7,10};
10. a.remove\_if(single\_digit());
12. list<**int**>::iterator it = a.begin();
13. **while**(it != a.end()){
14. cout<<\*it<<" ";
15. it++;
16. }

19. **return** 0;
20. }

输出：

10

版权声明：本文为博主原创文章，转载请标明原文地址，谢谢 ^\_^ https://blog.csdn.net/xiaoquantouer/article/details/70339869