* 没有退路可言、不要给自己找接口、以及顾虑太多
* 缺乏目的、目标、方法、效率再勤奋也只是一种时间的牺牲



@[toc]

# 常用排序算法

**学习目标：**

* 掌握常用的排序算法

**算法简介：**

* sort //对容器内元素进行排序
* random\_shuffle //洗牌 指定范围内的元素随机调整次序
* merge // 容器元素合并，并存储到另一容器中
* reverse // 反转指定范围的元素

## sort

**功能描述：**

* 对容器内元素进行排序

**函数原型：**

* sort(iterator beg, iterator end, \_Pred);
* // 按值查找元素，找到返回指定位置迭代器，找不到返回结束迭代器位置
* // beg 开始迭代器
* // end 结束迭代器
* // \_Pred 谓词

**示例：**

#include <algorithm>  
#include <vector>  
  
void myPrint(int val)  
{  
 cout << val << " ";  
}  
  
void test01() {  
 vector<int> v;  
 v.push\_back(10);  
 v.push\_back(30);  
 v.push\_back(50);  
 v.push\_back(20);  
 v.push\_back(40);  
  
 //sort默认从小到大排序  
 sort(v.begin(), v.end());  
 for\_each(v.begin(), v.end(), myPrint);  
 cout << endl;  
  
 //从大到小排序  
 sort(v.begin(), v.end(), greater<int>());  
 for\_each(v.begin(), v.end(), myPrint);  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结：**sort属于开发中最常用的算法之一，需熟练掌握

## random\_shuffle

**功能描述：**

* 洗牌 指定范围内的元素随机调整次序

**函数原型：**

* random\_shuffle(iterator beg, iterator end);
* // 指定范围内的元素随机调整次序
* // beg 开始迭代器
* // end 结束迭代器

**示例：**

#include <algorithm>  
#include <vector>  
#include <ctime>  
  
class myPrint  
{  
public:  
 void operator()(int val)  
 {  
 cout << val << " ";  
 }  
};  
  
void test01()  
{  
 srand((unsigned int)time(NULL));  
 vector<int> v;  
 for(int i = 0 ; i < 10;i++)  
 {  
 v.push\_back(i);  
 }  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
  
 //打乱顺序  
 random\_shuffle(v.begin(), v.end());  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结** : random\_shuffle洗牌算法比较实用，使用时记得加随机数种子

#include <ctime>  
srand((unsigned int)time(NULL));

## merge

**功能描述：**

* 两个容器元素合并，并存储到另一容器中

**函数原型：**

* merge(iterator beg1, iterator end1, iterator beg2, iterator end2, iterator dest);
* // 容器元素合并，并存储到另一容器中
* // 注意: 两个容器必须是**有序的**
* // beg1 容器1开始迭代器  
  // end1 容器1结束迭代器  
  // beg2 容器2开始迭代器  
  // end2 容器2结束迭代器  
  // dest 目标容器开始迭代器

**示例：**

#include <algorithm>  
#include <vector>  
  
class myPrint  
{  
public:  
 void operator()(int val)  
 {  
 cout << val << " ";  
 }  
};  
  
void test01()  
{  
 vector<int> v1;  
 vector<int> v2;  
 for (int i = 0; i < 10 ; i++)   
 {  
 v1.push\_back(i);  
 v2.push\_back(i + 1);  
 }  
  
 vector<int> vtarget;  
 //目标容器需要提前开辟空间  
 vtarget.resize(v1.size() + v2.size());  
 //合并 需要两个有序序列  
 merge(v1.begin(), v1.end(), v2.begin(), v2.end(), vtarget.begin());  
 for\_each(vtarget.begin(), vtarget.end(), myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结**: merge合并的两个容器必须的有序序列

## reverse

**功能描述：**

* 将容器内元素进行反转

**函数原型：**

* reverse(iterator beg, iterator end);
* // 反转指定范围的元素
* // beg 开始迭代器
* // end 结束迭代器

**示例：**

#include <algorithm>  
#include <vector>  
  
class myPrint  
{  
public:  
 void operator()(int val)  
 {  
 cout << val << " ";  
 }  
};  
  
void test01()  
{  
 vector<int> v;  
 v.push\_back(10);  
 v.push\_back(30);  
 v.push\_back(50);  
 v.push\_back(20);  
 v.push\_back(40);  
  
 cout << "反转前： " << endl;  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
  
 cout << "反转后： " << endl;  
  
 reverse(v.begin(), v.end());  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结：**reverse反转区间内元素，面试题可能涉及到

# 常用拷贝和替换算法

**学习目标：**

* 掌握常用的拷贝和替换算法

**算法简介：**

* copy // 容器内指定范围的元素拷贝到另一容器中
* replace // 将容器内指定范围的旧元素修改为新元素
* replace\_if // 容器内指定范围满足条件的元素替换为新元素
* swap // 互换两个容器的元素

## copy

**功能描述：**

* 容器内指定范围的元素拷贝到另一容器中

**函数原型：**

* copy(iterator beg, iterator end, iterator dest);
* // 按值查找元素，找到返回指定位置迭代器，找不到返回结束迭代器位置
* // beg 开始迭代器
* // end 结束迭代器
* // dest 目标起始迭代器

**示例：**

#include <algorithm>  
#include <vector>  
  
class myPrint  
{  
public:  
 void operator()(int val)  
 {  
 cout << val << " ";  
 }  
};  
  
void test01()  
{  
 vector<int> v1;  
 for (int i = 0; i < 10; i++) {  
 v1.push\_back(i + 1);  
 }  
 vector<int> v2;  
 v2.resize(v1.size());  
 copy(v1.begin(), v1.end(), v2.begin());  
  
 for\_each(v2.begin(), v2.end(), myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结** : 利用copy算法在拷贝时，目标容器记得提前开辟空间

## replace

**功能描述：**

* 将容器内指定范围的旧元素修改为新元素

**函数原型：**

* replace(iterator beg, iterator end, oldvalue, newvalue);
* // 将区间内旧元素 替换成 新元素
* // beg 开始迭代器
* // end 结束迭代器
* // oldvalue 旧元素
* // newvalue 新元素

**示例：**

#include <algorithm>  
#include <vector>  
  
class myPrint  
{  
public:  
 void operator()(int val)  
 {  
 cout << val << " ";  
 }  
};  
  
void test01()  
{  
 vector<int> v;  
 v.push\_back(20);  
 v.push\_back(30);  
 v.push\_back(20);  
 v.push\_back(40);  
 v.push\_back(50);  
 v.push\_back(10);  
 v.push\_back(20);  
  
 cout << "替换前：" << endl;  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
  
 //将容器中的20 替换成 2000  
 cout << "替换后：" << endl;  
 replace(v.begin(), v.end(), 20,2000);  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结** : replace会替换区间内满足条件的元素

## replace\_if

**功能描述:**

* 将区间内满足条件的元素，替换成指定元素

**函数原型：**

* replace\_if(iterator beg, iterator end, \_pred, newvalue);
* // 按条件替换元素，满足条件的替换成指定元素
* // beg 开始迭代器
* // end 结束迭代器
* // \_pred 谓词
* // newvalue 替换的新元素

**示例：**

#include <algorithm>  
#include <vector>  
  
class myPrint  
{  
public:  
 void operator()(int val)  
 {  
 cout << val << " ";  
 }  
};  
  
class ReplaceGreater30  
{  
public:  
 bool operator()(int val)  
 {  
 return val >= 30;  
 }  
  
};  
  
void test01()  
{  
 vector<int> v;  
 v.push\_back(20);  
 v.push\_back(30);  
 v.push\_back(20);  
 v.push\_back(40);  
 v.push\_back(50);  
 v.push\_back(10);  
 v.push\_back(20);  
  
 cout << "替换前：" << endl;  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
  
 //将容器中大于等于的30 替换成 3000  
 cout << "替换后：" << endl;  
 replace\_if(v.begin(), v.end(), ReplaceGreater30(), 3000);  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结** : replace\_if按条件查找，可以利用仿函数灵活筛选满足的条件

## swap

**功能描述：**

* 互换两个容器的元素

**函数原型：**

* swap(container c1, container c2);
* // 互换两个容器的元素
* // c1容器1
* // c2容器2

**示例：**

#include <algorithm>  
#include <vector>  
  
class myPrint  
{  
public:  
 void operator()(int val)  
 {  
 cout << val << " ";  
 }  
};  
  
void test01()  
{  
 vector<int> v1;  
 vector<int> v2;  
 for (int i = 0; i < 10; i++) {  
 v1.push\_back(i);  
 v2.push\_back(i+100);  
 }  
  
 cout << "交换前： " << endl;  
 for\_each(v1.begin(), v1.end(), myPrint());  
 cout << endl;  
 for\_each(v2.begin(), v2.end(), myPrint());  
 cout << endl;  
  
 cout << "交换后： " << endl;  
 swap(v1, v2);  
 for\_each(v1.begin(), v1.end(), myPrint());  
 cout << endl;  
 for\_each(v2.begin(), v2.end(), myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结** :swap交换容器时，注意交换的容器要同种类型