@[toc]

# 常用算术生成算法

**学习目标：**

* 掌握常用的算术生成算法

**注意：**

* 算术生成算法属于小型算法，使用时包含的头文件为 #include <numeric>

**算法简介：**

* accumulate // 计算容器元素累计总和
* fill // 向容器中添加元素

## accumulate

**功能描述：**

* 计算区间内 容器元素累计总和

**函数原型：**

* accumulate(iterator beg, iterator end, value);
* // 计算容器元素累计总和
* // beg 开始迭代器
* // end 结束迭代器
* // value 起始值

**示例：**

#include <numeric>  
#include <vector>  
void test01()  
{  
 vector<int> v;  
 for (int i = 0; i <= 100; i++) {  
 v.push\_back(i);  
 }  
  
 int total = accumulate(v.begin(), v.end(), 0);  
  
 cout << "total = " << total << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结：**accumulate使用时头文件注意是 numeric，这个算法很实用

## fill

**功能描述：**

* 向容器中填充指定的元素

**函数原型：**

* fill(iterator beg, iterator end, value);
* // 向容器中填充元素
* // beg 开始迭代器
* // end 结束迭代器
* // value 填充的值

**示例：**

#include <numeric>  
#include <vector>  
#include <algorithm>  
  
class myPrint  
{  
public:  
 void operator()(int val)  
 {  
 cout << val << " ";  
 }  
};  
  
void test01()  
{  
  
 vector<int> v;  
 v.resize(10);  
 //填充  
 fill(v.begin(), v.end(), 100);  
  
 for\_each(v.begin(), v.end(), myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结：**利用fill可以将容器区间内元素填充为 指定的值

# 常用集合算法

**学习目标：**

* 掌握常用的集合算法

**算法简介：**

* set\_intersection // 求两个容器的交集
* set\_union // 求两个容器的并集
* set\_difference // 求两个容器的差集

## set\_intersection

**功能描述：**

* 求两个容器的交集

**函数原型：**

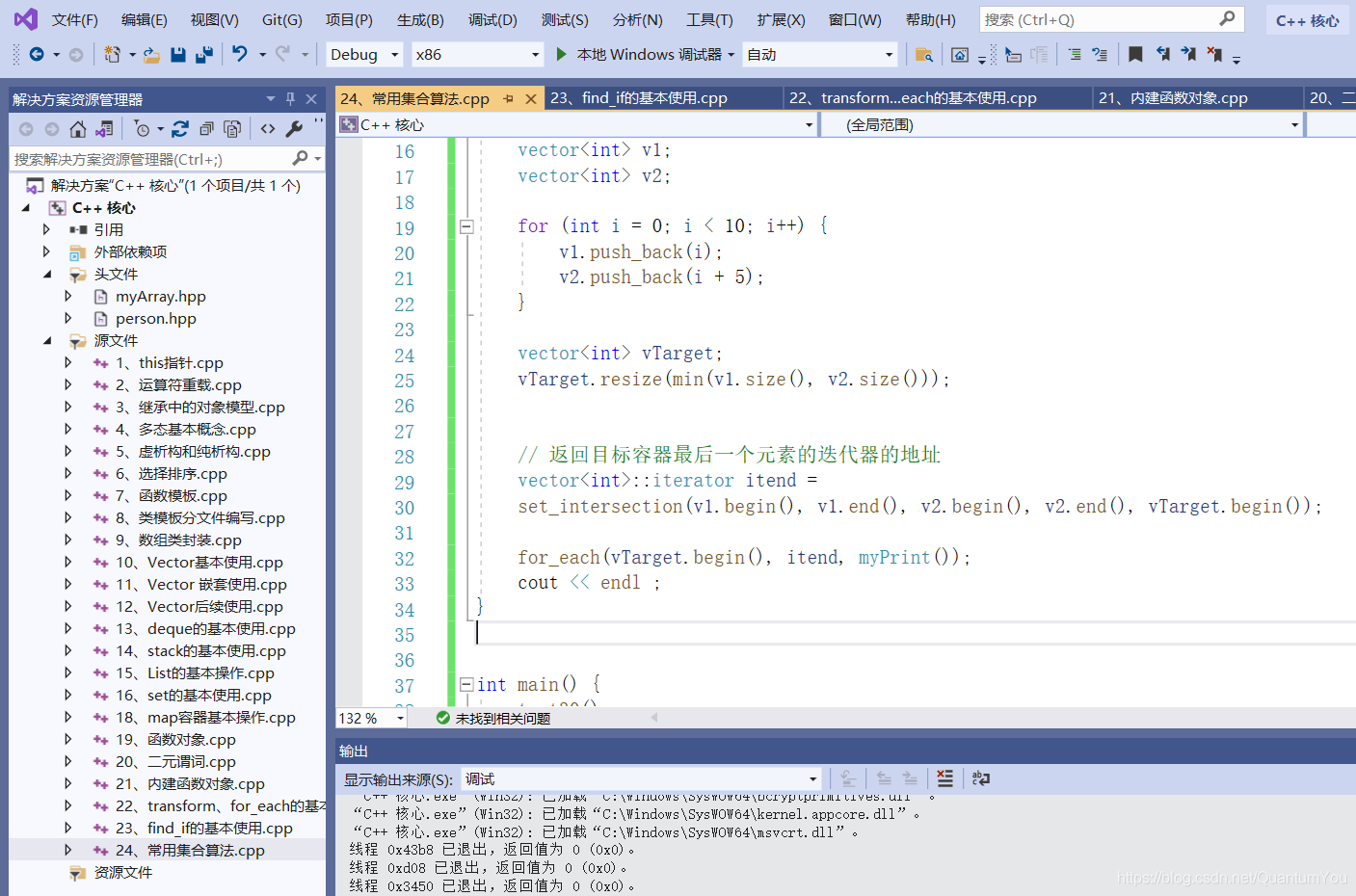
* set\_intersection(iterator beg1, iterator end1, iterator beg2, iterator end2, iterator dest);
* // 求两个集合的交集
* // **注意:两个集合必须是有序序列**
* // beg1 容器1开始迭代器  
  // end1 容器1结束迭代器  
  // beg2 容器2开始迭代器  
  // end2 容器2结束迭代器  
  // dest 目标容器开始迭代器

**示例：**

#include <vector>  
#include <algorithm>  
  
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+5);  
 }  
  
 vector<int> vTarget;  
 //取两个里面较小的值给目标容器开辟空间  
 vTarget.resize(min(v1.size(), v2.size()));  
  
 //返回目标容器的最后一个元素的迭代器地址  
 vector<int>::iterator itEnd =   
 set\_intersection(v1.begin(), v1.end(), v2.begin(), v2.end(), vTarget.begin());  
  
 for\_each(vTarget.begin(), itEnd, myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结：**

* 求交集的两个集合必须的有序序列
* 目标容器开辟空间需要从**两个容器中取小值**
* set\_intersection返回值既是交集中最后一个元素的位置



## set\_union

**功能描述：**

* 求两个集合的并集

**函数原型：**

* set\_union(iterator beg1, iterator end1, iterator beg2, iterator end2, iterator dest);
* // 求两个集合的并集
* // **注意:两个集合必须是有序序列**
* // 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 <vector>  
#include <algorithm>  
  
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+5);  
 }  
  
 vector<int> vTarget;  
 //取两个容器的和给目标容器开辟空间  
 vTarget.resize(v1.size() + v2.size());  
  
 //返回目标容器的最后一个元素的迭代器地址  
 vector<int>::iterator itEnd =   
 set\_union(v1.begin(), v1.end(), v2.begin(), v2.end(), vTarget.begin());  
  
 for\_each(vTarget.begin(), itEnd, myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结：**

* 求并集的两个集合必须的有序序列
* 目标容器开辟空间需要**两个容器相加**
* set\_union返回值既是并集中最后一个元素的位置

## set\_difference

**功能描述：**

* 求两个集合的差集

**函数原型：**

* set\_difference(iterator beg1, iterator end1, iterator beg2, iterator end2, iterator dest);
* // 求两个集合的差集
* // **注意:两个集合必须是有序序列**
* // 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 <vector>  
#include <algorithm>  
  
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+5);  
 }  
  
 vector<int> vTarget;  
 //取两个里面较大的值给目标容器开辟空间  
 vTarget.resize( max(v1.size() , v2.size()));  
  
 //返回目标容器的最后一个元素的迭代器地址  
 cout << "v1与v2的差集为： " << endl;  
 vector<int>::iterator itEnd =   
 set\_difference(v1.begin(), v1.end(), v2.begin(), v2.end(), vTarget.begin());  
 for\_each(vTarget.begin(), itEnd, myPrint());  
 cout << endl;  
  
  
 cout << "v2与v1的差集为： " << endl;  
 itEnd = set\_difference(v2.begin(), v2.end(), v1.begin(), v1.end(), vTarget.begin());  
 for\_each(vTarget.begin(), itEnd, myPrint());  
 cout << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结：**

* 求差集的两个集合必须的有序序列
* 目标容器开辟空间需要从**两个容器取较大值**
* set\_difference返回值既是差集中最后一个元素的位置