

知识点1【普通函数作为适配器】 ptr_fun

知识点2【常用的遍历算法】

- 1、for_each:遍历容器元素
 - 2、transform算法 将指定容器区间元素搬运到另一容器中
-

知识点3【常用的查找算法】

- 1、find算法 查找元素
 - 2、find_if条件查找
 - 3、adjacent_find算法 查找相邻重复元素
 - 4、binary_search算法 二分查找法（容器必须有序）
 - 5、count算法 统计元素出现次数
 - 6、count_if算法 统计元素出现次数
-

知识点4【常用排序算法】

- 1、merge算法 容器元素合并，并存储到另一容器中（每个容器必须有序）
 - 2、sort算法 容器元素排序
 - 3、random_shuffle算法 对指定范围内的元素随机调整次序
 - 4、reverse算法 反转指定范围的元素
-

知识点5【常用拷贝和替换算法】

- 1、copy算法 将容器内指定范围的元素拷贝到另一容器中
 - 2、replace算法 将容器内指定范围的旧元素修改为新元素
 - 3、replace_if算法 将容器内指定范围满足条件的元素替换为新元素
 - 4、swap算法 互换两个容器的元素
-

知识点6【算术生成算法】

1、accumulate算法 计算容器元素累计总和

2、fill算法 向容器中添加元素

知识点7【常用集合算法】

1、set_intersection算法 求两个set集合的交集

2、set_union算法 求两个set集合的并集

3、set_difference算法 求两个set集合的差集

知识点8【STL比赛练习】

知识点1【普通函数作为适配器】 *ptr_fun*

```
1 //普通函数 作为适配器
2 void myPrintInt01(int val,int tmp)
3 {
4     cout<<val+tmp<<" ";
5 }
6 void test01()
7 {
8     vector<int> v;
9     v.push_back(10);
10    v.push_back(20);
11    v.push_back(30);
12    v.push_back(40);
13
14    //普通函数 需要使用ptr_fun转换成函数适配器
15    for_each(v.begin(),v.end(), bind2nd(ptr_fun(myPrintInt01),1000));
16    cout<<endl;
17 }
```

运行结果:

C:\Qt\Qt5.8.0\tools\QtCreator\bin\qtcreator_process_stub.exe

1010 1020 1030 1040

知识点2 【常用的遍历算法】

1、for_each:遍历容器元素

4.3.1 for_each 遍历算法

```
/*  
    遍历算法 遍历容器元素  
    @param beg 开始迭代器  
    @param end 结束迭代器  
    @param _callback 函数回调或者函数对象  
    @return 函数对象  
*/  
for_each(iterator beg, iterator end, _callback);
```

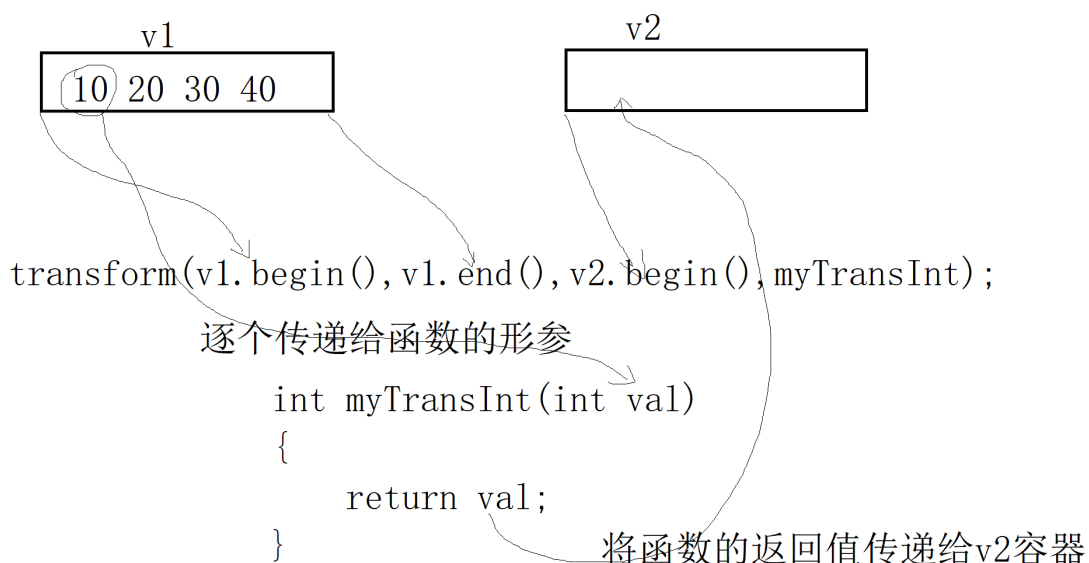
2、transform算法 将指定容器区间元素搬运到另一容器中

transform 算法 将指定容器区间元素搬运到另一容器中

注意：transform 不会给目标容器分配内存，所以需要我们提前分配好内存

```
@param beg1 源容器开始迭代器  
@param end1 源容器结束迭代器  
@param beg2 目标容器开始迭代器  
@param _callback 回调函数或者函数对象  
@return 返回目标容器迭代器  
*/  
transform(iterator beg1, iterator end1, iterator beg2, _callback);
```

案例1:



```

1  int myTransInt(int val)
2  {
3      return val;
4  }
5  class MyTransInt{
6  public:
7      int operator()(int val)
8      {
9          return val;
10     }
11 };
12
13 void test02()
14 {
15     vector<int> v1;
16     v1.push_back(10);
17     v1.push_back(20);
18     v1.push_back(30);
19     v1.push_back(40);
20
21     //将v1容器的元素 帮运到 v2中
22     vector<int> v2;
23     //预先: 设置v2的大小 (注意!!!)
24     v2.resize(v1.size());
25
26     //transform(v1.begin(),v1.end(),v2.begin(), 搬运方式);
27     //transform(v1.begin(),v1.end(),v2.begin(),myTransInt);
28     transform(v1.begin(),v1.end(),v2.begin(),MyTransInt());
29
30     for_each(v2.begin(),v2.end(),[](int val){cout<<val<<" ";});
31     cout<<endl;
32 }

```

运行结果:

10 20 30 40

知识点3 【常用的查找算法】

1、find算法 查找元素

/*

find 算法 查找元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param value 查找的元素

@return 返回查找元素的位置

*/

find(iterator beg, iterator end, value)

失败返回v.end()

案例:

```
1 #include<string>
2 class Person
3 {
4 public:
5     string name;
6     int age;
7 public:
8     Person(string name,int age)
9     {
10         this->name = name;
11         this->age = age;
12     }
13     bool operator==(const Person &ob)
14     {
```

```

15  if(this->name == ob.name && this->age==ob.age)
16  return true;
17  return false;
18  }
19  };
20  void test03()
21  {
22  vector<int> v1;
23  v1.push_back(10);
24  v1.push_back(20);
25  v1.push_back(30);
26  v1.push_back(40);
27
28  vector<int>::iterator ret;
29  ret = find(v1.begin(),v1.end(),20);
30  if(ret != v1.end())
31  {
32  cout<<"找到的数据为:"<<*ret<<endl;
33  }
34
35  vector<Person> v2;
36  v2.push_back(Person("德玛西亚",18));
37  v2.push_back(Person("小法",19));
38  v2.push_back(Person("小炮",20));
39  v2.push_back(Person("牛头",21));
40
41  Person tmp("小炮",20);
42  vector<Person>::iterator ret2;
43  //对于find寻找自定义数据 需要重载==
44  ret2 = find(v2.begin(),v2.end(),tmp);
45  if(ret2 != v2.end())
46  {
47  cout<<"找到的数据name="<<(*ret2).name<<",age="<<(*ret2).age<<endl;
48  }
49
50  }

```

运行结果：

找到的数据为:20
找到的数据name=小炮, age=20

2、find_if条件查找

```
/*
    find_if 算法 条件查找
    @param beg 容器开始迭代器
    @param end 容器结束迭代器
    @param callback 回调函数或者谓词(返回 bool 类型的函数对象)
    @return bool 查找返回true 否则false
*/
find_if(iterator beg, iterator end, _callback);
```

```
1 bool myGreaterThan20(int val)
2 {
3     return val>20;
4 }
5 class MyGreaterThan20{
6 public:
7     bool operator()(int val)
8     {
9         return val>20;
10    }
11 };
12
13 void test04()
14 {
15     vector<int> v1;
16     v1.push_back(10);
17     v1.push_back(20);
18     v1.push_back(30);
19     v1.push_back(40);
20
21     //寻找第一个大于20的数
22     vector<int>::iterator ret;
23     //ret = find_if(v1.begin(),v1.end(), myGreaterThan20 );
```

```

24  ret = find_if(v1.begin(),v1.end(), MyGreaterThan20() );
25  if(ret != v1.end())
26  {
27      cout<<"寻找到的数据为:"<<*ret<<endl;
28  }
29  }

```

运行结果:

 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

寻找到的数据为:30

3、adjacent_find算法 查找相邻重复元素

```

/*
    adjacent_find 算法 查找相邻重复元素
    @param beg 容器开始迭代器
    @param end 容器结束迭代器
    @param _callback 回调函数或者谓词( 返回 bool 类型的函数对象)
    @return 返回相邻元素的第一个位置的迭代器
*/
adjacent_find(iterator beg, iterator| end, _callback);

```

```

1  void test05()
2  {
3      vector<int> v1;
4      v1.push_back(10);
5      v1.push_back(20);
6      v1.push_back(20);
7      v1.push_back(40);
8      v1.push_back(50);
9      v1.push_back(50);
10     vector<int>::iterator ret;
11     //对于普通数据 不需要回调函数
12     ret = adjacent_find(v1.begin(),v1.end());
13     if(ret != v1.end())
14     {
15         cout<<"寻找重复的数据:"<<*ret<<endl;
16     }

```



```

17
18 vector<Person> v2;
19 v2.push_back(Person("德玛西亚",18));
20 v2.push_back(Person("小法",19));
21 v2.push_back(Person("小法",19));
22 v2.push_back(Person("牛头",21));
23 vector<Person>::iterator ret2;
24 ret2 = adjacent_find(v2.begin(),v2.end());
25 if(ret2 != v2.end())
26 {
27 cout<<"寻找到重复的数据:"<<\
28 (*ret2).name<<" "<<(*ret2).age<<endl;
29 }
30 }

```

运行结果:

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

寻找到重复的数据:20
寻找到重复的数据:小法 19

4、binary_search算法 二分查找法 (容器必须有序)

/*

binary_search 算法 二分查找法

注意: 在无序序列中不可用

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param value 查找的元素

@return bool 查找返回 true 否则 false

*/

bool binary_search(iterator beg, iterator end, value);

案例:

```

1 void test06()
2 {
3     vector<int> v1;
4     v1.push_back(10);
5     v1.push_back(20);


```

```

6  v1.push_back(30);
7  v1.push_back(40);
8  v1.push_back(50);
9
10 bool ret = binary_search(v1.begin(),v1.end(),30);
11 if(ret == true)
12 {
13     cout<<"找到"<<endl;
14 }
15 else
16 {
17     cout<<"未找到"<<endl;
18 }
19 }

```

运行结果:

 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcra



5、*count*算法 统计元素出现次数

/*

count 算法 统计元素出现次数

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param value 回调函数或者谓词(返回 *bool* 类型的函数对象)

@return int 返回元素个数

*/

count(iterator beg, iterator end, value);

案例:

```

1 void test07()
2 {

```

```

3  vector<int> v1;
4  v1.push_back(10);
5  v1.push_back(20);
6  v1.push_back(10);
7  v1.push_back(40);
8  v1.push_back(10);
9
10  cout<<count(v1.begin(),v1.end(),10)<<endl;//3
11  }

```

6、*count_if* 算法 统计元素出现次数

count_if 算法 统计元素出现次数

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param callback 回调函数或者谓词(返回 bool 类型的函数对象)

@return int 返回元素个数

*/

```
count_if(iterator beg, iterator end, _callback);
```

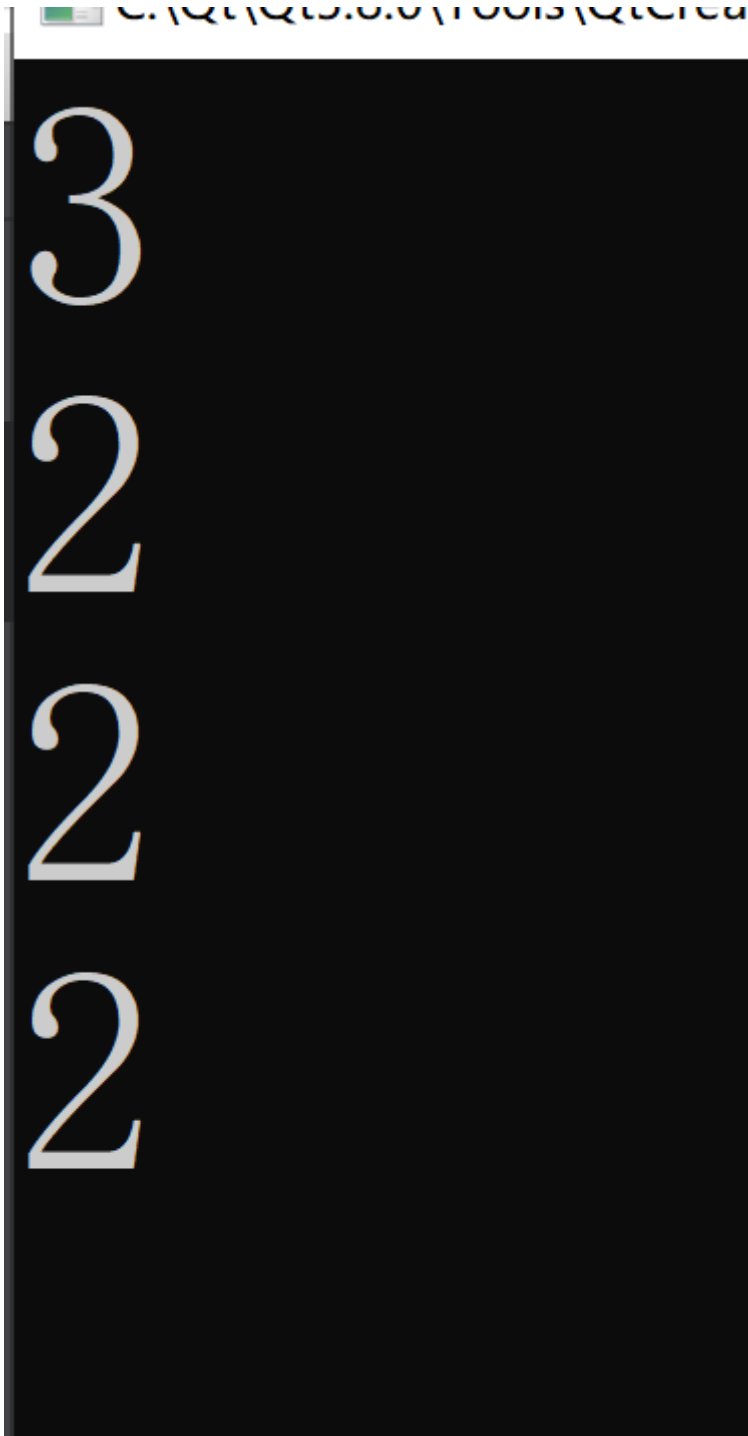
```

1  bool myGreaterThan10(int val)
2  {
3      return val>10;
4  }
5  class MyGreaterThan10
6  {
7  public:
8      bool operator()(int val)
9      {
10         return val>10;
11     }
12 };
13 void test07()
14 {
15     vector<int> v1;
16     v1.push_back(10);
17     v1.push_back(20);
18     v1.push_back(10);
19     v1.push_back(40);
20     v1.push_back(10);
21
22     cout<<count(v1.begin(),v1.end(),10)<<endl;//3

```

```
23
24 //普通函数
25 cout<<count_if(v1.begin(),v1.end(), myGreaterThanOrEqualTo10)<<endl;//2
26 //函数对象
27 cout<<count_if(v1.begin(),v1.end(), MyGreaterThanOrEqualTo10())<<endl;//2
28 //内建函数对象
29 cout<<count_if(v1.begin(),v1.end(), bind2nd(greater<int>(),10))
<<endl;//2
30 }
```

运行结果:



知识点4 【常用排序算法】

1、merge算法 容器元素合并，并存储到另一容器中（每个容器必须有顺序）

```
/*
    merge 算法 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
    @param beg1 容器1 开始迭代器
    @param end1 容器1 结束迭代器
    @param beg2 容器2 开始迭代器
    @param end2 容器2 结束迭代器
    @param dest 目标容器开始迭代器
*/
merge(iterator beg1, iterator end1, iterator beg2, iterator end2, itera
tor dest)
```

案例：

```
1 void test01()
2 {
3     vector<int> v1;
4     v1.push_back(1);
5     v1.push_back(3);
6     v1.push_back(5);
7     v1.push_back(7);
8
9     vector<int> v2;
10    v2.push_back(2);
11    v2.push_back(4);
12    v2.push_back(6);
13    v2.push_back(8);
14
15    vector<int> v3;
16    //预先：设置v3的大小
17    v3.resize(v1.size()+v2.size());
18
19    merge(v1.begin(),v1.end(),v2.begin(),v2.end(),v3.begin());
20
21    for_each(v3.begin(),v3.end(),[](int val){cout<<val<<" ";});
22    cout<<endl;
23 }
```

运行结果：

1 2 3 4 5 6 7 8

2、*sort*算法 容器元素排序

```
/*
    sort 算法 容器元素排序
    @param beg 容器1 开始迭代器
    @param end 容器1 结束迭代器
    @param _callback 回调函数或者谓词( 返回 bool 类型的函数对象)
*/
sort(iterator beg, iterator end, _callback)
```


3、*random_shuffle*算法 对指定范围内的元素随机调整次序

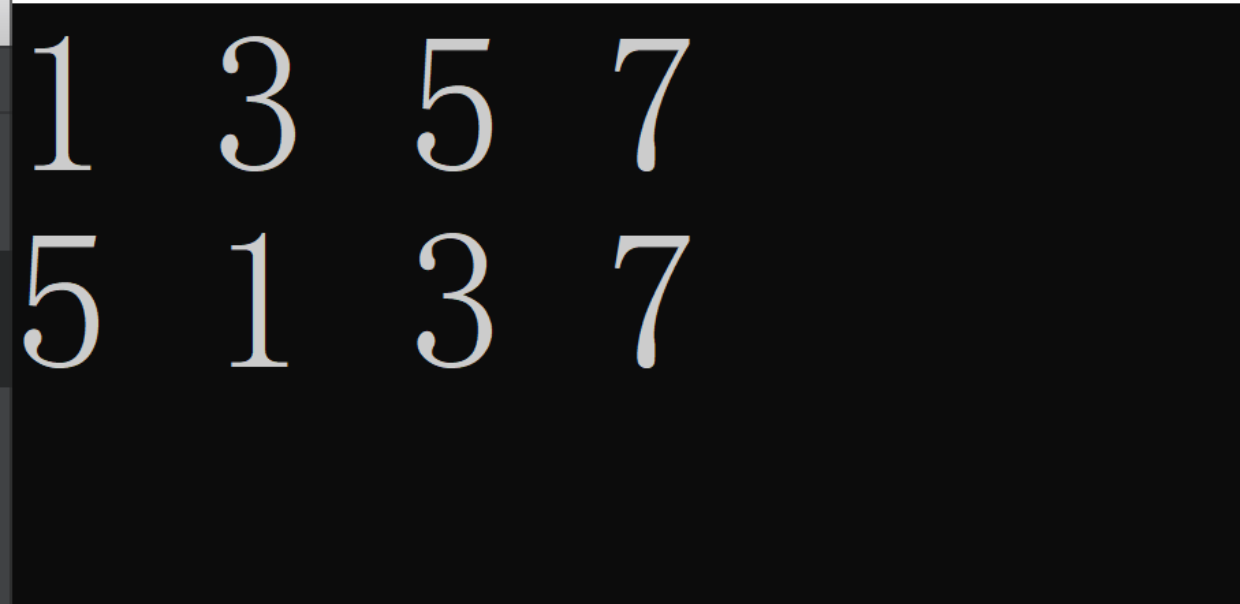
```
/*
    random_shuffle 算法 对指定范围内的元素随机调整次序
    @param beg 容器开始迭代器
    @param end 容器结束迭代器
*/
random_shuffle(iterator beg, iterator end)
```

```
1 #include<time.h>
2 void test02()
3 {
4     vector<int> v1;
5     v1.push_back(1);
6     v1.push_back(3);
7     v1.push_back(5);
8     v1.push_back(7);
9
10    //srand设置种子
11    srand(time(NULL));
12
13    for_each(v1.begin(),v1.end(),[](int val){cout<<val<<" ";});
14    cout<<endl;
15
16    //需要配置 srand
17    random_shuffle(v1.begin(),v1.end());
18
19    for_each(v1.begin(),v1.end(),[](int val){cout<<val<<" ";});
```

```
20  cout<<endl;
21  }
```

运行结果：

 C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe



4、*reverse*算法 反转指定范围的元素

/*

reverse 算法 反转指定范围的元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

*/

reverse(iterator beg, iterator end)

案例：

```
1  void test03()
2  {
3      vector<int> v1;
4      v1.push_back(1);
5      v1.push_back(3);
6      v1.push_back(5);
7      v1.push_back(7);
8      for_each(v1.begin(),v1.end(),[](int val){cout<<val<<" ";});
9      cout<<endl;//1 3 5 7
10
11     reverse(v1.begin(),v1.end());
12
```

```

13  for_each(v1.begin(),v1.end(),[](int val){cout<<val<<" ";});
14  cout<<endl;//7 5 3 1
15  }

```

知识点5 【常用拷贝和替换算法】

1、*copy*算法 将容器内指定范围的元素拷贝到另一容器中

```

/*
    copy 算法 将容器内指定范围的元素拷贝到另一容器中
    @param beg 容器开始迭代器
    @param end 容器结束迭代器
    @param dest 目标起始迭代器
*/
copy(iterator beg, iterator end, iterator dest)

```

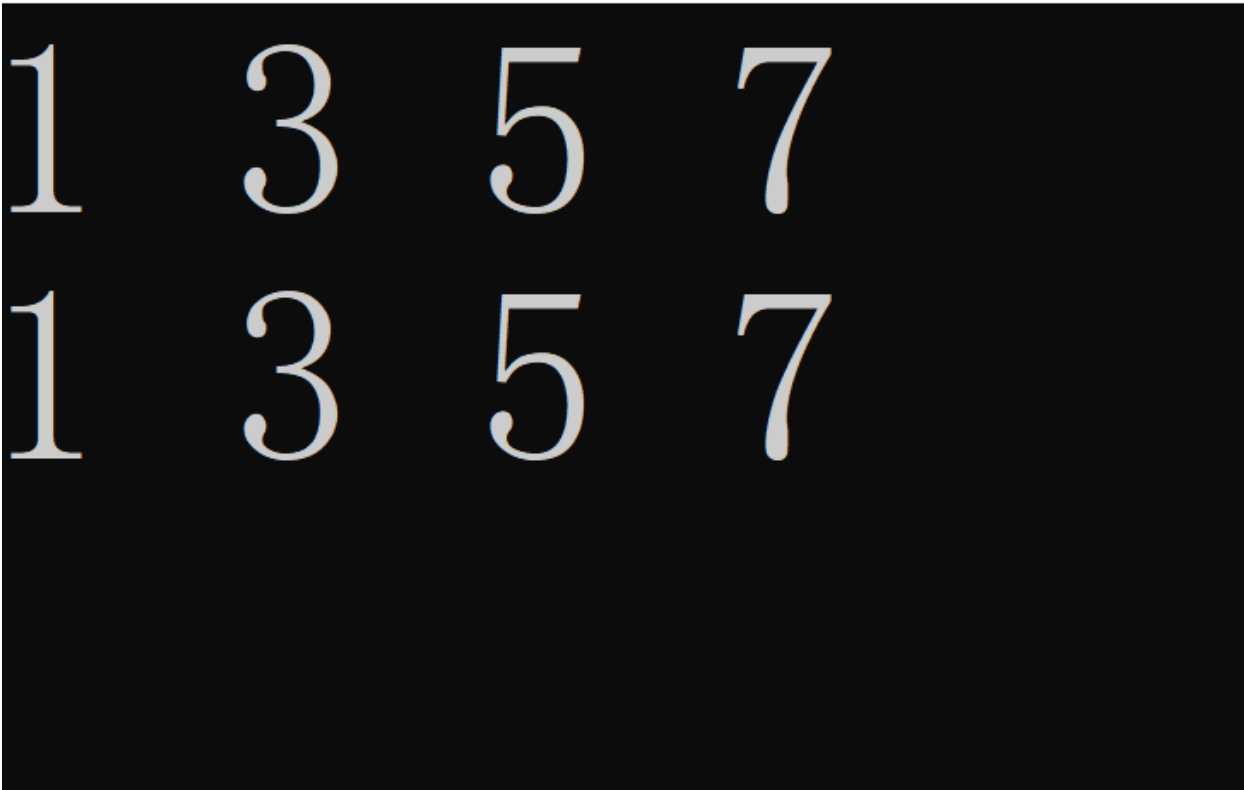
案例：

```

1  #include<iterator>
2  void test04()
3  {
4      vector<int> v1;
5      v1.push_back(1);
6      v1.push_back(3);
7      v1.push_back(5);
8      v1.push_back(7);
9
10     vector<int> v2;
11     //预先：设置大小
12     v2.resize(v1.size());
13
14     copy(v1.begin(),v1.end(),v2.begin());
15
16     for_each(v2.begin(),v2.end(),[](int val){cout<<val<<" ";});
17     cout<<endl;//1 3 5 7
18
19     //copy秀一下：用copy输出(了解)
20     copy(v2.begin(),v2.end(), ostream_iterator<int>(cout," ") );
21 }

```

运行结果：



2、*replace*算法 将容器内指定范围的旧元素修改为新元素

```
/*
```

```
    replace 算法 将容器内指定范围的旧元素修改为新元素
```

```
    @param beg 容器开始迭代器
```

```
    @param end 容器结束迭代器
```

```
    @param oldvalue 旧元素
```

```
    @param oldvalue 新元素
```

```
*/
```

```
replace(iterator beg, iterator end, oldvalue, newvalue)
```

案例：

3、*replace_if*算法 将容器内指定范围满足条件的元素替换为新元素

```
/*
```

```
    replace_if 算法 将容器内指定范围满足条件的元素替换为新元素
```

```
    @param beg 容器开始迭代器
```

```
    @param end 容器结束迭代器
```

```
    @param callback 函数回调或者谓词( 返回 Bool 类型的函数对象)
```

```
    @param oldvalue 新元素
```

```
*/
```

```
replace_if(iterator beg, iterator end, _callback, newvalue)
```

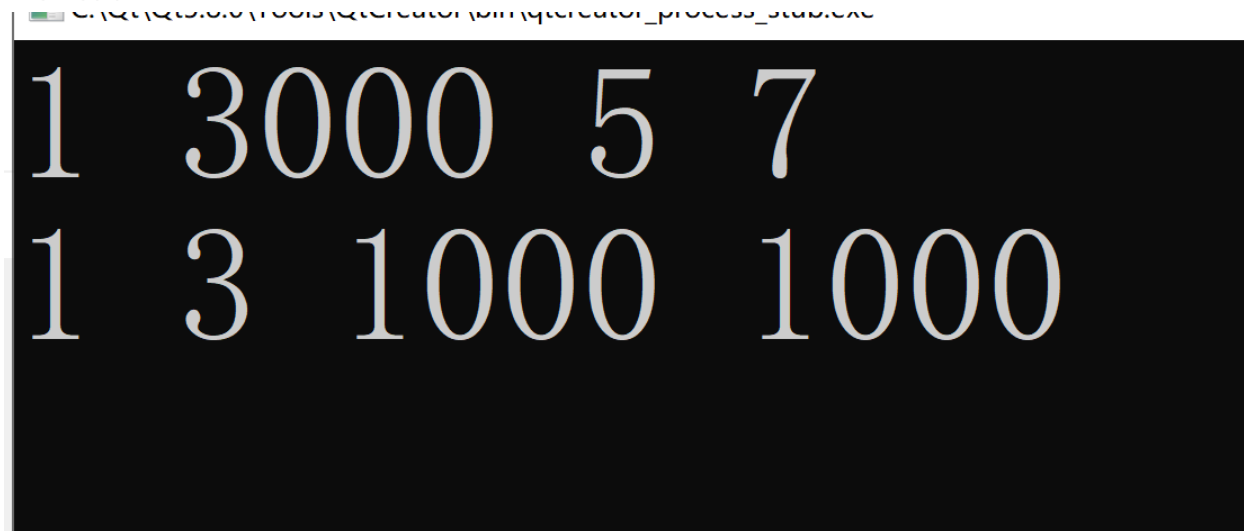
```
1 void test05()  
2 {  
3     vector<int> v1;
```

```

4  v1.push_back(1);
5  v1.push_back(3);
6  v1.push_back(5);
7  v1.push_back(7);
8
9  //将容器中的3替换成3000
10 replace(v1.begin(),v1.end(),3,3000);
11 copy(v1.begin(),v1.end(), ostream_iterator<int>(cout," ") );
12 cout<<endl;
13
14 vector<int> v2;
15 v2.push_back(1);
16 v2.push_back(3);
17 v2.push_back(5);
18 v2.push_back(7);
19 //将容器的大于3替换成3000
20 replace_if(v2.begin(),v2.end(), bind2nd(greater<int>(),3) , 1000 );
21 copy(v2.begin(),v2.end(), ostream_iterator<int>(cout," ") );
22 cout<<endl;
23 }

```

运行结果：



```

C:\Qt\Qt5.9.6\tools\qtcreator\bin\qtcreator_process_stable.exe
1 3000 5 7
1 3 1000 1000

```

4、*swap*算法 互换两个容器的元素

```
/*  
    swap 算法 互换两个容器的元素  
    @param c1 容器 1  
    @param c2 容器 2  
*/  
swap(container c1, container c2)|
```

知识点6 【算术生成算法】

1、*accumulate*算法 计算容器元素累计总和

```
/*  
    accumulate 算法 计算容器元素累计总和  
    @param beg 容器开始迭代器  
    @param end 容器结束迭代器  
    @param value 累加值  
*/  
accumulate(iterator beg, iterator end, value)
```

案例:

```
1 void test06()  
2 {  
3     vector<int> v1;  
4     v1.push_back(1);  
5     v1.push_back(3);  
6     v1.push_back(5);  
7     v1.push_back(7);  
8  
9     int sum = accumulate(v1.begin(),v1.end(),1000);  
10    cout<<"sum = "<<sum<<endl;//1016  
11 }
```

2、*fill*算法 向容器中添加元素

/*

fill 算法 向容器中添加元素

@param beg 容器开始迭代器

@param end 容器结束迭代器

@param value t 填充元素

*/

fill(iterator beg, iterator end, value)

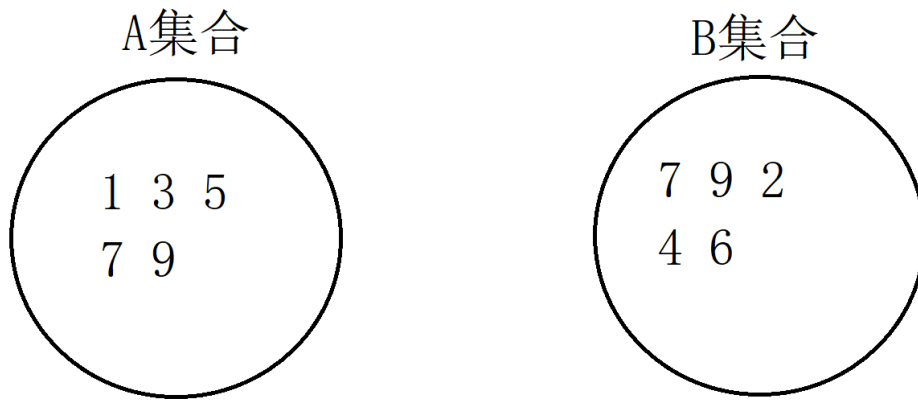
案例:

```
1 #include<iterator>
2 void test07()
3 {
4     vector<int> v;
5     v.resize(5);
6
7     fill(v.begin(),v.end(), 100);
8
9     copy(v.begin(),v.end(), ostream_iterator<int>(cout," ") );
10    cout<<endl;
11 }
12 int main(int argc, char* argv[])
```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

100 100 100 100 100

知识点7 【常用集合算法】



A 和 B 并集: 1 3 5 7 9 2 4 6

A 和 B 交集: 7 9

A 差 B: 1 3 5

B 差 A: 2 4 6

1、*set_intersection*算法 求两个set集合的交集

```
1  /*
2  set_intersection算法 求两个set集合的交集
3  注意:两个集合必须是有序序列
4  @param beg1 容器1开始迭代器
5  @param end1 容器1结束迭代器
6  @param beg2 容器2开始迭代器
7  @param end2 容器2结束迭代器
8  @param dest 目标容器开始迭代器
9  @return 目标容器的最后一个元素的迭代器地址
10 */
11 set_intersection(iterator beg1, iterator end1, iterator beg2, iterator end2, iterator dest)
```

2、*set_union*算法 求两个set集合的并集

```
1  /*
2  set_union算法 求两个set集合的并集
3  注意:两个集合必须是有序序列
4  @param beg1 容器1开始迭代器
5  @param end1 容器1结束迭代器
6  @param beg2 容器2开始迭代器
7  @param end2 容器2结束迭代器
8  @param dest 目标容器开始迭代器
```

```

9  @return 目标容器的最后一个元素的迭代器地址
10 */
11 set_union(iterator beg1, iterator end1, iterator beg2, iterator end2, it
erator dest)

```

3、*set_difference*算法 求两个set集合的差集

```

1  /*
2  set_difference算法 求两个set集合的差集
3  注意:两个集合必须是有序序列
4  @param beg1 容器1开始迭代器
5  @param end1 容器1结束迭代器
6  @param beg2 容器2开始迭代器
7  @param end2 容器2结束迭代器
8  @param dest 目标容器开始迭代器
9  @return 目标容器的最后一个元素的迭代器地址
10 */
11 set_difference(iterator beg1, iterator end1, iterator beg2, iterator end
2, iterator dest)

```

案例:

```

1  void test08()
2  {
3      vector<int> A;
4      A.push_back(1);
5      A.push_back(3);
6      A.push_back(5);
7      A.push_back(7);
8      A.push_back(9);
9
10     vector<int> B;
11     B.push_back(7);
12     B.push_back(9);
13     B.push_back(2);
14     B.push_back(4);
15     B.push_back(6);
16
17     //求交集
18     vector<int> v1;//存放交集
19     v1.resize(min(A.size(),B.size()));
20
21     vector<int>::iterator myEnd=
set_intersection(A.begin(),A.end(),B.begin(),B.end(), v1.begin());

```

```

22  copy(v1.begin(), myEnd, ostream_iterator<int>(cout, " ")) );
23  cout<<endl;
24
25  //求并集
26  vector<int> v2;//存放并集
27  v2.resize(A.size()+B.size() );
28  myEnd = set_union(A.begin(),A.end(),B.begin(),B.end(), v2.begin());
29  copy(v2.begin(), myEnd, ostream_iterator<int>(cout, " ")) );
30  cout<<endl;
31
32  //求差集 A 差 B
33  vector<int> v3;//存放并集
34  v3.resize( A.size() );
35  myEnd = set_difference(A.begin(),A.end(),B.begin(),B.end(),
v3.begin());
36  copy(v3.begin(), myEnd, ostream_iterator<int>(cout, " ")) );
37  cout<<endl;
38  }

```

运行结果：

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator_process_stub.exe

```

7 9
1 3 5 7 9 2 4 6
1 3 5

```

知识点8 【STL比赛练习】

比赛规则：

某市举行一场演讲比赛（speech_contest），共有24个人参加。比赛共三轮，前两轮为淘汰赛，第三轮为决赛。

比赛方式：分组比赛，每组6个人；选手每次要随机分组，进行比赛；

第一轮分为4个小组，每组6个人。比如编号为: 100-123. 整体进行抽签

(draw) 后顺序演讲。当小组演讲完后，淘汰组内排名最后的三个选手，然后继续下一个小组的比赛。

第二轮分为2个小组，每组6人。比赛完毕，淘汰组内排名最后的三个选手，然后继续下一个小组的比赛。

第三轮只剩下1组6个人，本轮为决赛，选出前三名。 比赛评分：10个评委打分，去除最低、最高分，求平均分每个选手演讲完由10个评委分别打分。该选手的最终得分是去掉一个最高分和一个最低分，求得剩下的8个成绩的平均分。选手的名次按得分降序排列。

用STL编程，求解这个问题 1) 请打印出所有选手的名字与参赛号，并以参赛号的升序排列。 2) 打印每一轮比赛后，小组比赛成绩和小组晋级名单

需求分析： 1) 产生选手 (ABCDEFGHIJKLMNOPQRSTUVWXYZ) 姓名、得分；选手编号

第1轮 选手抽签 选手比赛 查看比赛结果

第2轮 选手抽签 选手比赛 查看比赛结果

第3轮 选手抽签 选手比赛 查看比赛结果

实现思路： 需要把选手信息、选手得分信息、选手比赛抽签信息、选手的晋级信息保存在容器中，需要涉及到各个容器的选型。 选手可以设计一个类 Speaker (姓名和得分) 所有选手的编号可以单独放在一个vector容器中，做抽签用

所有选手编号和选手信息，可以放在容器内

容器说明：

选手：

```
1 class Speaker
2 {
3 public:
4     string name;
5     int score[3];
6 public:
7     Speaker()
8     {
9     ;
10 }
```



```

11  Speaker(string name)
12  {
13      this->name=name;
14      memset(score,0,sizeof(score));
15  }
16  };

```

编号容器：

```

1  vector<int>

```

<编号、选手> map容器

```

1  map<int, Speaker> m;//<编号,选手>

```

每一组的分数容器<分数，编号，排序规则> multimap容器

```

1  multimap<int,int, greater<int>>> m2;

```

```

1  #include <iostream>
2  #include<string>
3  #include<unistd.h>
4  #include<vector>
5  #include<map>
6  #include<algorithm>
7  using namespace std;
8  class Speaker
9  {
10 public:
11     string name;
12     int score[3];
13 public:
14     Speaker()
15     {
16     ;
17     }
18     Speaker(string name)
19     {
20         this->name=name;
21         memset(score,0,sizeof(score));
22     }

```

```

23 };
24 void createSpeaker(vector<int> &v, map<int,Speaker> &m)
25 {
26     string tmp="ABCDEFGHJKLMNOPQRSTUVWXYZ";
27     for(int i=0;i<24;i++)
28     {
29         //存放选手编号
30         v.push_back(100+i);
31
32         //存放《编号,选手》
33         string name = "选手";
34         name += tmp[i];
35         m.insert(make_pair(i+100, Speaker(name)));
36     }
37 }
38 #include<time.h>
39 #include<deque>
40 void speech_contest(int index,vector<int> &v, map<int,Speaker> &m, vector<int> &v1);
41 void printSpeechResault(int index,vector<int> &v, map<int,Speaker> &m, vector<int> &v1);
42 int main(int argc, char *argv[])
43 {
44     //创建24名选手、将选手<编号,选手>放入map容器中 选手编号放vector<>容器中
45     vector<int> v;//选手编号
46     map<int, Speaker> m;//<编号,选手>
47
48     //创建选手24名
49     createSpeaker(v, m);
50
51     //设置种子
52     srand(time(NULL));
53
54     //第一轮：参加的选手抽签
55     random_shuffle(v.begin(),v.end());
56     //进行第一轮比赛
57     //1表示当前轮数 v选手编号 m选手信息 v1晋级容器
58     vector<int> v1;
59     speech_contest(1, v, m, v1);
60     //打印第一轮比赛结果：所有参与比赛的成绩 晋级的名单
61     printSpeechResault(1, v, m, v1);

```

```

62
63 //第二轮比赛：参加的选手抽签
64 random_shuffle(v1.begin(),v1.end());
65 //进行第二轮比赛
66 //1表示当前轮数 v选手编号 m选手信息 v1晋级容器
67 vector<int> v2;
68 speech_contest(2, v1, m, v2);
69 //打印第一轮比赛结果：所有参与比赛的成绩 晋级的名单
70 printSpeechResault(2, v1, m, v2);
71
72 //第二轮比赛：参加的选手抽签
73 random_shuffle(v2.begin(),v2.end());
74 //进行第二轮比赛
75 //1表示当前轮数 v选手编号 m选手信息 v1晋级容器
76 vector<int> v3;
77 speech_contest(3, v2, m, v3);
78 //打印第一轮比赛结果：所有参与比赛的成绩 晋级的名单
79 printSpeechResault(3, v2, m, v3);
80
81 return 0;
82 }
83 #if 1
84 void printSpeechResault(int index,vector<int> &v, map<int,Speaker> &m,
vector<int> &v1)
85 {
86 cout<<"第"<<index<<"轮比赛成绩如下"<<endl;
87 int count = 0;
88 int n = 0;
89 vector<int>::iterator mit=v1.begin();
90 for(vector<int>::iterator it=v.begin(); it!=v.end(); it++)
91 {
92 if(count%6 == 0)
93 {
94 n = count/6+1;
95 cout<<"第"<<n<<"组的成绩如下:"<<endl;
96 }
97 count++;
98
99 cout<<"姓名:"<<m[*it].name<<"， 得分:"<<m[*it].score[index-1]<<endl;
100
101 //每个组的成绩打印完 立马打印晋级名单

```

```

102  if(count%6 == 0)
103  {
104      cout<<"第"<<n<<"组的晋级名单如下:"<<endl;
105      for(int i=0;i<3;i++,mit++)
106      {
107          cout<<"姓名:"<<m[*mit].name<<" , 得分:"<<m[*mit].score[index-1]<<endl;
108      }
109  }
110  }
111  }
112  #endif
113  #if 1
114  void speech_contest(int index,vector<int> &v, map<int,Speaker> &m, vector<int> &v1)
115  {
116      int count= 0;
117      //设计一个<分数,编号>的map容器 存放每一组的《分数, 编号》
118      multimap<int,int, greater<int>> m2;
119
120      //选手逐一登台比赛
121      for(vector<int>::iterator it=v.begin();it!=v.end();it++)
122      {
123          count++;
124          /*it 代表一名选手编号
125          //10个评委打分
126          deque<int> d;
127          for(int i=0;i<10;i++)
128          {
129              int score = rand()%41+60;//60~100
130              d.push_back(score);
131          }
132          //对d容器排序
133          sort(d.begin(),d.end());
134          //去掉最高分 以及最低分
135          d.pop_back();
136          d.pop_front();
137          //求取总分数
138          int sum = accumulate(d.begin(),d.end(),0);
139          //求平均分 并赋值给选手的score[index-1]
140          //map<int,Speaker>

```

```
141 int avg = sum/d.size();
142 (m[*it]).score[index-1] = avg;
143
144 m2.insert(make_pair(avg, *it));
145
146 if(count % 6 == 0)//刚好已经有6人 把上面的6人的成绩取前3
147 {
148     //90 80 70 60 50 40
149     //遍历m2容器 取出前3名的 编号
150     int i=0;
151     for(multimap<int,int, greater<int>>::iterator mit=m2.begin();\
152         mit!=m2.end() && i<3 ;mit++,i++)
153     {
154         //将晋级的编号 放入晋级的容器v1中
155         v1.push_back( (*mit).second );
156     }
157
158     //将上一组的m2清空
159     m2.clear();//清空
160 }
161 }
162 }
163 #endif
164
```