# Sort函数使用小结

函数名 功能描述    
sort 对给定区间所有元素进行排序    
stable\_sort 对给定区间所有元素进行稳定排序    
partial\_sort 对给定区间所有元素部分排序    
partial\_sort\_copy 对给定区间复制并排序    
nth\_element 找出给定区间的某个位置对应的元素    
is\_sorted 判断一个区间是否已经排好序    
partition 使得符合某个条件的元素放在前面    
stable\_partition 相对稳定的使得符合某个条件的元素放在前面    
  
要使用此函数只需用#include <algorithm> sort即可使用，语法描述为：  
sort(begin,end)，表示一个范围，例如：  
int \_tmain(int argc, \_TCHAR\* argv[])  
{  
 int a[20]={2,4,1,23,5,76,0,43,24,65},i;  
 for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
 sort(a,a+20);  
 for(i=0;i<20;i++)  
 cout<<a[i]<<endl;  
 return 0;  
}  
输出结果将是把数组a按升序排序，说到这里可能就有人会问怎么样用它降序排列呢？这就是下一个讨论的内容.  
一种是自己编写一个比较函数来实现，接着调用三个参数的sort：sort(begin,end,compare)就成了。对于list容器，这个方法也适用，把compare作为sort的参数就可以了，即：sort(compare).  
  
1）自己编写compare函数：  
bool compare(int a,int b)  
{  
  return a<b; //升序排列，如果改为return a>b，则为降序  
}  
int \_tmain(int argc, \_TCHAR\* argv[])  
{  
  int a[20]={2,4,1,23,5,76,0,43,24,65},i;  
  for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
  sort(a,a+20,compare);  
  for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
  return 0;  
}  
  
2）更进一步，让这种操作更加能适应变化。也就是说，能给比较函数一个参数，用来指示是按升序还是按降序排,这回轮到函数对象出场了。  
为了描述方便，我先定义一个枚举类型EnumComp用来表示升序和降序。很简单：  
enum Enumcomp{ASC,DESC};  
然后开始用一个类来描述这个函数对象。它会根据它的参数来决定是采用“<”还是“>”。  
  
class compare  
{  
  private:  
  Enumcomp comp;  
  public:  
  compare(Enumcomp c):comp(c) {};  
  bool operator () (int num1,int num2)  
  {  
  switch(comp)  
  {  
  case ASC:  
  return num1<num2;  
  case DESC:  
  return num1>num2;  
  }  
  }  
};  
  
接下来使用 sort(begin,end,compare(ASC)实现升序，sort(begin,end,compare(DESC)实现降序。  
主函数为：  
int main()  
{  
  int a[20]={2,4,1,23,5,76,0,43,24,65},i;  
  for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
  sort(a,a+20,compare(DESC));  
  for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
  return 0;  
}  
3)其实对于这么简单的任务（类型支持“<”、“>”等比较运算符），完全没必要自己写一个类出来。标准库里已经有现成的了，就在functional里，include进来就行了。functional提供了一堆基于模板的比较函数对象。它们是（看名字就知道意思了）：equal\_to<Type>、not\_equal\_to<Type>、greater<Type>、greater\_equal<Type>、less<Type>、less\_equal<Type>。对于这个问题来说，greater和less就足够了，直接拿过来用：  
  
升序：sort(begin,end,less<data-type>());  
降序：sort(begin,end,greater<data-type>()).  
int \_tmain(int argc, \_TCHAR\* argv[])  
{  
  int a[20]={2,4,1,23,5,76,0,43,24,65},i;  
  for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
  sort(a,a+20,greater<int>());  
  for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
  return 0;  
}  
4)既然有迭代器，如果是string 就可以使用反向迭代器来完成逆序排列，程序如下：  
int main()  
{  
  string str("cvicses");  
  string s(str.rbegin(),str.rend());  
  cout << s <<endl;  
  return 0;  
}  
qsort():  
原型:  
\_CRTIMP void \_\_cdecl qsort (void\*, size\_t, size\_t,int (\*)(const void\*, const void\*));  
解释: qsort ( 数组名 ，元素个数，元素占用的空间(sizeof)，比较函数)  
比较函数是一个自己写的函数 遵循 int com(const void \*a,const void \*b) 的格式。  
当a b关系为 > < = 时，分别返回正值 负值 零 （或者相反）。  
使用a b 时要强制转换类型，从void \* 转换回应有的类型后，进行操作。  
数组下标从零开始,个数为N, 下标0-(n-1)。  
实例：  
int compare(const void \*a,const void \*b)  
{  
  return \*(int\*)b-\*(int\*)a;     
}  
int main()  
{  
  int a[20]={2,4,1,23,5,76,0,43,24,65},i;  
  for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
  qsort((void \*)a,20,sizeof(int),compare);  
  for(i=0;i<20;i++)  
  cout<<a[i]<<endl;  
  return 0;  
}  
相关:  
1)why你必须给予元素个数？  
因为阵列不知道它自己有多少个元素  
2)why你必须给予大小？  
因为 qsort 不知道它要排序的单位.  
3)why你必须写那个丑陋的、用来比较俩数值的函式？  
因为 qsort 需要一个指标指向某个函式，因为它不知道它所要排序的元素型别.  
4)why qsort 所使用的比较函式接受的是 const void\* 引数而不是 char\* 引数？  
因为 qsort 可以对非字串的数值排序.

二维字符数组排序：

#include <iostream>

#include <cstring>

#include <algorithm>

using namespace std;

struct Data

{

char data[100];

}str[100];

bool cmp(const Data &elem1, const Data &elem2)

{

if (strcmp(elem1.data, elem2.data) < 0)

return true;

return false;

}

int main()

{

int n, i;

while (cin>>n)

{

for (i=0; i<n; ++i)

{

cin>>str[i].data;

}

sort(str, str+n, cmp);

for (i=0; i<n; ++i)

cout<<str[i].data<<endl;

}

return 0;

}