Sort二级排序

#include <iostream>

#include <map>

#include <cstdio>

#include <algorithm>

#include <vector>

using namespace std;

typedef pair<string,int>P;

typedef vector<P>V;

typedef map<string,int>M;

**bool cmp(P a,P b)**

**{**

**if(a.second>b.second)**

**return a.second>b.second;**

**else if(a.second==b.second)**

**{**

**return a.first<b.first;**

**}**

**return false;**

**}**

V v;

int main()

{

string color;

int n;

M m;

freopen("input1.txt","r",stdin);

while(cin>>n)

{

getchar();

for(int i=1;i<=n;i++)

{

getline(cin,color);

P p;

p.first=color;

p.second=m[color]++;

v.push\_back(p);

}

sort(v.begin(), v.end(), cmp);

cout<<v.front().first<<endl;

int num=v.front().second;

for(V::iterator it=v.begin()+1;it!=v.end();it++)

{

if((\*it).second<num) break;

cout<<(\*it).first<<endl;

}

}

return 0;

}

qsort（基本快速排序的方法，每次把数组分成两部分和中间的一个划分值，而对于有多个重复值的数组来说，基本快速排序的效率较低，且不稳定）。集成在C语言库函数里面的的qsort函数，使用 三 路划分的方法解决排序这个问题。所谓三路划分，是指把数组划分成小于划分值，等于划分值和大于划分值的三个部分。

具体介绍：-^^

void qsort( void \*base, size\_t num, size\_t width, int (\_\_cdecl \*compare )

int compare (const void \*elem1, const void \*elem2 ) );

qsort（即，quicksort）主要根据你给的比较条件给一个快速排序，主要是通过指针移动实现排序功能。排序之后的结果仍然放在原来数组中。

参数意义如下:

第一个参数 base 是 需要排序的目标数组名（或者也可以理解成开始排序的地址，因为可以写&s[i]这样的表达式）

第二个参数 num 是 参与排序的目标数组元素个数

第三个参数 width 是单个元素的大小（或者目标数组中每一个元素长度），推荐使用sizeof(s[0]）这样的表达式

第四个参数 compare 就是让很多人觉得非常困惑的比较函数啦。

我们来简单讨论compare这个比较函数（写成compare是我的个人喜好，你可以随便写成什么，比如 cmp 什么的，在后面我会一直用cmp做解释）。

典型的compare的定义是int compare(const void \*a,const void \*b);

返回值必须是int，两个参数的类型必须都是const void \*，那个a,b是随便写的，个人喜好。假设是对int排序的话，如果是升序，那么就是如果a比b大返回一个正值，小则负值，相等返回0，其他的依次类推，后面有例子来说明对不同的类型如何进行排序。

qsort 的使用方法：

一、对int类型数组排序

int num[100];

int cmp ( const void \*a , const void \*b )

{

return \*(int \*)a - \*(int \*)b; //升序排序

//return \*(int \*)b - \*(int \*)a; //降序排序

/\*可见：参数列表是两个空指针，现在他要去指向你的数组元素。所以转型为你当前的类型，然后取值。

升序排列时，若第一个参数指针指向的“值”大于第二个参数指针指向的“值”，则返回正；若第一个参数指针指向的“值”等于第二个参数指针指向的“值”，则返回零；若第一个参数指针指向的“值”小于第二个参数指针指向的“值”，则返回负。

降序排列时，则刚好相反。

\*/

}

qsort(s,n,sizeof(s[0]),cmp);

示例完整函数（已在 VC6.0上运行通过）：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

int s[10000],n,i;

int cmp(const void \*a,const void \*b)

{

return(\*(int \*)b-\*(int \*)a); //实现的是升序排序

}

int main()

{

// 输入想要输入的数的个数

scanf("%d",&n);

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

scanf("%d",&s[i]);

qsort(s,n,sizeof(s[0]),cmp);

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

printf("%d ",s[i]);

return(0);

}

二、对char类型数组排序（同int类型）

char word[100];

int cmp( const void \*a , const void \*b )

{

//注意，网上很多版本是 “ return \*(char \*)a - \*(int \*)b; ”

//因为编辑者的不用心，盲目copy，以讹传讹，传的一直是错的 \*(int \*)b

//应该是return \*(char \*)a - \*(char \*)b;

return \*(char \*)a - \*(char \*)b;

}

qsort(word,100,sizeof(word[0]),cmp);

//附，可能 getchar(); 会派上用场

三、对double类型数组排序（特别要注意）

double in[100];

int cmp( const void \*a , const void \*b )

{

return \*(double \*)a > \*(double \*)b ? 1 : -1;

//返回值的问题，显然cmp返回的是一个整型，所以避免double返回小数而被丢失，用一个判断返回值。

}

qsort(in,100,sizeof(in[0]),cmp);

//附：排序结果的输出，一般建议用 “ %g ” 格式

/\* 在这里多嘴一句，"%g"格式输出 虽然书上是说系统会自动选择 " %f " 格式 和 " %e " 格式 中长度较短的格式，并去掉无意义的0，但实际上系统如果选择了" %e "，系统会输出比 “ %e " 格式更省一位的格式输出。（此结论，来自VC6.0的实际操作）\*/

四、对结构体一级排序

struct In

{

double data;

int other;

}s[100]

//按照data的值从小到大将结构体排序,关于结构体内的排序关键数据data的类型可以很多种，参考上面的例子写

int cmp( const void \*a ,const void \*b)

{

return (\*(In \*)a).data > (\*(In \*)b).data ? 1 : -1;

//注意，这条语句在VC6.0环境下运行可能会出错，但是并不是语句错了，而是你要先 Build ，或者全部重建。总之语句是对的。

//或者你可以将这上面1条语句改成下面这3条语句

//struct In \*aa = (In \*)a;

//struct In \*bb = (In \*)b;

//return aa->data > bb->data ? 1 : -1;

}

qsort(s,100,sizeof(s[0]),cmp);

五、对结构体二级排序

struct In

{

int x; //你可以比喻成：失败次数

int y; //你可以比喻成：成功次数

}s[100];

//按照x从小到大排序，当x相等时按照y从大到小排序。 你可以想象成：失败是主要因素的一个问题，先比较 失败次数少，失败次数相同 再看 成功次数多。

int cmp( const void \*a , const void \*b )

{

struct In \*c = (In \*)a;

struct In \*d = (In \*)b;

if(c->x != d->x) return c->x - d->x;

else return d->y - c->y;

}

qsort(s,100,sizeof(s[0]),cmp);

六、对字符串进行排序

struct In

{

int data;

char str[100];

}s[100];

//按照结构体中字符串str的字典顺序排序

int cmp ( const void \*a , const void \*b )

{

return strcmp( (\*(In \*)a)->str , (\*(In \*)b)->str );

}

qsort(s,100,sizeof(s[0]),cmp);

注意！qsort 中的 cmp 得自己写 。

再说说 sort （常用于 C++ ）

sort 使用时得注明：using namespace std; 或直接打 std::sort() 还得加上 #include <algorithm> 头文件

例：

#include<iostream>

#include<algorithm>

using namespace std;

int main()

{

int a[20];

　　 for(int i=0;i<20;++i)

cin>>a[i];

　 sort(a,a+20); //范围，很明显这里是a+20 注意，这是必要的，如果是a+19

for(i=0;i<20;i++) //最后一个值a[19]就不会参与排序。

cout<<a[i]<<endl;

return 0;

}

std::sort是一个改进版的qsort. std::sort函数优于qsort的一些特点：对大数组采取9项取样，更完全的三路划分算法，更细致的对不同数组大小采用不同方法排序。

最后，我们来说说sort、qsort的区别：

sort是qsort的升级版，如果能用sort尽量用sort，使用也比较简单，不像qsort还得自己去写 cmp 函数，只要注明 使用的库函数就可以使用，参数只有两个（如果是普通用法）头指针和尾指针；

默认sort排序后是升序，如果想让他降序排列，可以使用自己编的cmp函数

#include<iostream>

#include<algorithm>

using namespace std;

int cmp(int a,int b)

{

if(a<b)

return 1; //升序排列，如果改为 a >b，则为降序,要注意sort()中cmp()的返值只有1和0，不像qsort中存在－1！！！！

else

return 0;

}

int main(){

int i;

int a[20];

for(int i=0;i<5;++i)

cin>>a[i];

sort(a,a+5,cmp); //范围，很明显这里是a+5 注意，这是必要的，如果是a+4最后一个值a[4]就不会参与排序。

for(i=0;i<5;i++)

cout<<a[i]<<endl;

system("pause");

return 0;

}

对二维数组的排序：

#include <iostream>

#include <algorithm>

#include <ctime>

using namespace std;

bool cmp(int \*p,int \*q)

{

if(p[0]==q[0])

{

if(p[1]==q[1])

{

return p[2]<q[2];

}

else return p[1]<q[1];

}

else return p[0]<q[0];

}

int main()

{

srand(time(0));

int i;

int \*\*a=new int\*[1000];

for(i=0;i<1000;++i)

{

a[i]=new int[3];

a[i][0]=rand()%1000;

a[i][1]=rand()%1000;

a[i][2]=rand()%1000;

//printf("%d\t%d\t%d\n",a[i][0],a[i][1],a[i][2]);

}

sort(a,a+1000,cmp);

/\*cout<<"After sort"<<endl;

for(i=0;i<1000;++i)

{

printf("%d\t%d\t%d\n",a[i][0],a[i][1],a[i][2]);

}\*/

return 0;

}