qsort函数使用方法

qsort函数是一个标准C库函数，提供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。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。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。

qsort包含在<stdlib.h>头文件中，此函数根据你给的比较条件进行快速排序，通过指针移动实现排序。排序之后的结果仍然放在原数组中。使用qsort函数必须自己写一个比较函数。

函数原型为：

void qsort ( void \* base, size\_t num, size\_t size, int ( \* comparator ) ( const void \*, const void \* ) );

函数一共四个参数，没返回值。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 size 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 comparator就是让很多人觉得非常困惑的比较函数啦。这个在后面说明。

一个典型的qsort的写法如下:

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

其中第一个参数是参与排序的数组名（或者也可以理解成开始排序的地址，因为可以写&s[i]，这个问题下面有说明）；第二个参数是参与排序的元素个数；第三个参数是单个元素的大小（推荐使用sizeof(s[0])这样的表达式，下面也有说明）；第四个参数就是很多人觉得非常困惑的比较函数，关于这个函数,还要说的比较麻烦...

下面来讨论cmp这个比较函数（写成cmp是我的个人喜好,你可以随便写成什么，比如qcmp什么的）。典型的cmp的定义是:

int cmp(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是我随便写的两个参数名。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，后面有例子来说明对不同的类型如何进行排序。

在函数体内要对a，b进行强制类型转换后才能得到正确的返回值，不同的类型有不同的处理方法。具体情况请参考后面的例子。

\*\*\*\* 关于快排的一些小问题 \*\*\*\*

1、快排是不稳定的，这个不稳定一个表现在其使用的时间是不确定的，最好情况(O(n))和最坏情况(O(n^2))差距太大，我们一般说的O(nlog(n))都是指的是其平均时间。

2、快排是不稳定的，这个不稳定表现在如果相同的比较元素，可能顺序不一样，假设我们有这样一个序列,3,3,3,但是这三个3是有区别的，我们标记为3a,3b,3c,快排后的结果不一定就是3a,3b,3c这样的排列，所以在某些特定场合我们要用结构体来使其稳定（No.6的例子就是说明这个问题的）。

3、快排的比较函数的两个参数必须都是const void \*的,这个要特别注意，写a和b只是我的个人喜好，写成cmp也只是我的个人喜好。推荐在cmp里面重新定义两个指针来强制类型转换，特别是在对结构体进行排序的时候。

4、快排qsort的第三个参数，那个sizeof，推荐是使用sizeof(s[0])这样，特别是对结构体，往往自己定义2\*sizeof(int)这样的会出问题，用sizeof(s[0])既方便又保险。

5、如果要对数组进行部分排序，比如对一个s[n]的数组排列其从s[i]开始的m个元素，只需要在第一个和第二个参数上进行一些修改:

void qsort(&s[i],m,sizeof(s[i]),cmp);

\*\*\* 标程，举例说明 \*\*\*

No.1、手工实现QuickSort:

基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

#include<stdio.h>

#include<stdlib.h>

void QuickSort(int \*A,int left,int right)

{

if(left>=right) return;

int x=A[(left+right)>>1],low=left,high=right;

while(low<high)

{

while(A[low]<x)

low++;

while(A[high]>x)

high--;

if(low<=high)

{

int Temp=A[low];

A[low]=A[high];

A[high]=Temp;

low++;

high--;

}

}

QuickSort(A,left,high);

QuickSort(A,low,right);

}

int main()

{

int length,i,r[10000];

scanf("%d",&length);

for(i=1;i<=length;i++)

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

QuickSort(r,1,length);

for(i=1;i<=length;i++)

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

printf("\n");

system("pause");

return 0;

}

或者：

#include<stdio.h>

#include<stdlib.h>

int QKPass(int r[],int left,int right)

{

int low,high,x;

x=r[left];//选择基准记录

low=left;

high=right;

while(low<high)

{

while(low<high&&r[high]>=x)//从右到左找到小于x的记录

high--;

if(low<high)

{

r[low]=r[high];//找到后进行替换

low++;

}

while(low<high&&r[low]<x)

{

low++;

}

if(low<high)

{

r[high]=r[low];

high--;

}

r[low]=x;//将基准记录保存在low=high的位置

return low;//返回基准记录的位置

}

}

int QKSort(int r[],int low,int high)

{

int pos;

if(low<high)

{

pos=QKPass(r,low,high);

QKSort(r,low,pos-1);

QKSort(r,pos+1,high);

}

}

int main()

{

int length,i,r[10000];

scanf("%d",&length);

for(i=1;i<=length;i++)

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

QKSort(r,1,length);

for(i=1;i<=length;i++)

printf("%-3d",r[i]);

printf("\n");

system("pause");

return 0;

}

No.2、最常见的，对int数组排序:

#include<stdio.h>

#include<stdlib.h>

int s[10000],n,i;

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

{

return(\*(int \*)a-\*(int \*)b); //升序

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

}

int main()

{

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]);

printf("\n");

system("pause");

return 0;

}

No.3、对double型数组排序，原理同int：

这里做个注释，本来是因为要判断如果a==b返回0的，但是严格来说，两个double数是不可能相等的，只能说fabs(a-b)<1e-20之类的这样来判断，所以这里只返回了1和-1

#include<stdio.h>

#include<stdlib.h>

double s[1000];

int i,n;

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

{

return((\*(double\*)a-\*(double\*)b>0)?1:-1);

}

int main()

{

scanf("%d",&n);

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

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

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

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

printf("%.2lf ",s[i]);

printf("\n");

system("pause");

return(0);

}

No.4、对一个字符数组排序.原理同int:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char s[10000],i,n;

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

{

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

}

int main()

{

scanf("%s",s);

n=strlen(s);

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

printf("%s",s);

printf("\n");

system("pause");

return 0;

}

No.5、对结构体排序（一级排序）:

很多时候我们都会对结构体排序，比如2010年校赛的那个根据几个参数排序，一般这个时候都在cmp函数里面先强制转换了类型，不要在return里面转换，我也说不清为什么，但是这样程序会更清晰，并且绝对是没错的。 这里同样请注意double返回0的问题:

#include<stdio.h>

#include<stdlib.h>

struct node

{

double data;

int no;

} s[100];

int i,n;

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

{

struct node \*aa=(node \*)a;

struct node \*bb=(node \*)b;

return(((aa->data)>(bb->data))?1:-1);

}

int main()

{

scanf("%d",&n);

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

{

s[i].no=i+1;

scanf("%lf",&s[i].data);

}

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

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

printf("%d %lf\n",s[i].no,s[i].data);

system("pause");

return 0;

}

No.6、对结构体排序（二级排序）。加入no来使其稳定（即data值相等的情况下按原来的顺序排）:

[复制代码](javascript:void(0);)

#include<stdio.h>

#include<stdlib.h>

struct node

{

double data;

int no;

} s[100];

int i,n;

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

{

struct node \*aa=(node \*)a;

struct node \*bb=(node \*)b;

if(aa->data!=bb->data)

return(((aa->data)>(bb->data))?1:-1);

else

return((aa->no)-(bb->no));

}

int main()

{

scanf("%d",&n);

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

{

s[i].no=i+1;

scanf("%lf",&s[i].data);

}

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

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

printf("%d %lf\n",s[i].no,s[i].data);

printf("\n");

system("pause");

return 0;

}

如果有字符串的话，就这样写：

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

{

struct node \*aa=(node \*)a;

struct node \*bb=(node \*)b;

if(aa->data!=bb->data)

return(((aa->data)>(bb->data))?1:-1);

else

return((aa->no)-(bb->no));

else

return strcmp(aa.str,bb.str);

//return strcmp(aa->str,bb->str);

//按照结构体中字符串str的字典顺序排序

}

No.7、对字符串数组的排序（char s[][]型）:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char s[100][100];

int i,n;

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

{

return(strcmp((char\*)a,(char\*)b));

}

int main()

{

scanf("%d",&n);

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

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

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

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

printf("%s\n",s[i]);

printf("\n");

system("pause");

return 0;

}

No.8、对字符串数组排序（char \*s[]型）:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char \*s[100];

int i,n;

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

{

return(strcmp(\*(char\*\*)a,\*(char\*\*)b));

}

int main()

{

scanf("%d",&n);

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

{

s[i]=(char\*)malloc(sizeof(char\*));

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

}

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

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

printf("%s\n",s[i]);

printf("\n");

system("pause");

return 0;

}

**qsort 的使用方法的一些示例：**

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); //实现的是降序排序  
}  
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 。

参考链接：https://www.cnblogs.com/ForeverJoker/archive/2013/05/25/qsort-sort.html

**用Qsort排序链表的使用实例记录**

原创 2015年01月18日 21:58:55

设计思路：要排序的是链表，排序结构体的时候本身创建的就是结构体数组所以空间连续可排。但是链表是离散的，不能直接Qsort。所以分配一个临时空间用来存储地址，然后排序地址再重新建立。

之所以记录在这里是因为其中写cmp函数的时候，发现形参怎么写还是有些讲究的，于是放在这里以方便查看！

**//--------------------------------struct部分**

**[cpp]** [view plain](http://blog.csdn.net/u011497904/article/details/42844289) [copy](http://blog.csdn.net/u011497904/article/details/42844289)

1. **typedef** **struct** LinkedlistNode//链表节点
2. {
3. **int** data;
4. LinkedlistNode\* next;
5. }node;
7. **struct** LinkInfo //用来记录链表头结点、最后一个节点、元素个数
8. {
9. **int** listlen;
10. LinkedlistNode \*head,\*tail;
11. };

**//--------------------------Qsort部分**

1. cmpIncrease(**const** **void** \*left,**const** **void** \*right)
2. {
3. //node \*\*p=(node \*\*)left;
4. node \*p=\*(node \*\*)left;//cmp函数两种访问方式
5. node \*\*q=(node \*\*)right;
6. **if**(p->data>(\*q)->data)
7. **return** 1;
8. **else**
9. **return** p->data==(\*q)->data?0:-1;
10. }
11. **int** cmpDecrease(**const** **void** \*left,**const** **void** \*right)
12. {
13. //node \*\*p=(node \*\*)left;
14. node \*p=\*(node \*\*)left;//cmp函数两种访问方式
15. node \*\*q=(node \*\*)right;
16. **if**(p->data<(\*q)->data)
17. **return** 1;
18. **else**
19. **return** p->data==(\*q)->data?0:-1;
20. }
22. **void** LinkSortQ(LinkInfo \*L,**bool** increase)
23. {
24. **if**(L->listlen<=1)
25. **return**;
26. node\* (\*all)=**new** node\* [L->listlen];
27. all[0]=L->head->next;
28. **for**(**int** i=1;i<L->listlen;i++)
29. all[i]=(all[i-1])->next;
30. qsort((node \*)(all),L->listlen,**sizeof**(node \*),increase?cmpIncrease:cmpDecrease);
31. node \*start=L->head;
32. **for**(**int** i=0;i<L->listlen;i++)
33. {
34. start->next=all[i];
35. start=start->next;
36. }
37. L->tail=all[L->listlen-1];
38. **delete**(all);
39. }

参考链接：http://blog.csdn.net/u011497904/article/details/42844289