



北京大学
PEKING UNIVERSITY

信息科学技术学院

程序设计与算法(一)

李文新 郭炜



指针(二)

指针和二维数组

如果定义二维数组：

$T \ a[M][N];$

➤ $a[i]$ (i 是整数) 是一个一维数组

指针和二维数组

如果定义二维数组：

$T \ a[M][N];$

➤ $a[i]$ (i 是整数) 是一个一维数组

➤ $a[i]$ 的类型是 $T *$

指针和二维数组

如果定义二维数组：

$T \ a[M][N];$

➤ $a[i]$ (i 是整数) 是一个一维数组

➤ $a[i]$ 的类型是 $T *$

➤ $\text{sizeof}(a[i]) = \text{sizeof}(T) * N$

指针和二维数组

如果定义二维数组：

$T \ a[M][N];$

➤ $a[i]$ (i 是整数) 是一个一维数组

➤ $a[i]$ 的类型是 $T *$

➤ $\text{sizeof}(a[i]) = \text{sizeof}(T) * N$

➤ $a[i]$ 指向的地址： 数组 a 的起始地址 + $i \times N \times \text{sizeof}(T)$

指针和二维数组

```
void Reverse(int * p,int size) { //颠倒一个数组
    for(int i = 0;i < size/2; ++i) {
        int tmp = p[i];
        p[i] = p[size-1-i];
        p[size-1-i] = tmp;
    }
}

int a[3][4] = { {1,2,3,4},{5,6,7,8},{9,10,11,12}};

Reverse(a[1],4);

=> { {1,2,3,4},{8,7,6,5},{9,10,11,12}};
```

指针和二维数组

```
void Reverse(int * p,int size) { //颠倒一个数组
    for(int i = 0;i < size/2; ++i) {
        int tmp = p[i];
        p[i] = p[size-1-i];
        p[size-1-i] = tmp;
    }
}

int a[3][4] = { {1,2,3,4},{5,6,7,8},{9,10,11,12}};

Reverse(a[1],4);
=> { {1,2,3,4},{8,7,6,5},{9,10,11,12}};
Reverse(a[1],6);
=> { {1,2,3,4},{10,9,5,6},{7,8,11,12}};
```


指向指针的指针

定义:

$T^{**} p;$

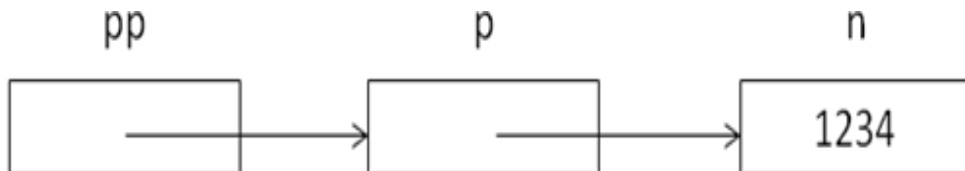
p 是指向指针的指针, p 指向的地方应该存放着一个类型为 T^* 的指针

$*p$ 的类型是 T^*

指向指针的指针

```
#include <iostream>
using namespace std;
int main()
{
    int **pp; //指向int*类型指针的指针
    int * p;
    int n = 1234;
    p = &n;    // p指向n
    pp = &p; //pp指向p
    cout << *(*pp) << endl; // *pp是p, 所以*(*pp)就是n
    return 0;
}
```

=> 1234



指针和字符串

- 字符串常量的类型就是 `char *`
- 字符数组名的类型也是 `char *`

```
#include <iostream>
using namespace std;
int main()
{
    char * p = "Please input your name:\n";
    cout << p ; // 若不用cout, printf(p) 亦可
    char name[20];
    char * pName = name;
    cin >> pName;
    cout << "Your name is " << pName;
    return 0;
}
```

```
Please input your name:
Jack✓
Your name is Jack
```

指针和字符串

- 字符数组名的类型也是 `char *`，就是一个地址

```
char name[20];
```

```
int n;
```

```
scanf("%d%s", &n, name);
```

```
cin >> n >> name;
```

字符串操作库函数

函数名称	功能
strcat	将一个字符串连接到另一个字符串后面
strchr	查找某字符在字符串中最先出现的位置
strrchr	查找某字符在字符串中最后出现的位置
strstr	求子串的位置
strcmp	比较两个字符串的大小（大小写相关）
stricmp	比较两个字符串的大小（大小写无关）
strcpy	字符串拷贝
strlen	求字符串长度
strlwr	将字符串变成小写
strupr	将字符串变成大写
strncat	将一个字符串的前n个字符连接到另一个字符串后面
strncmp	比较两个字符串的前n个字符
strncpy	拷贝字符串的前n个字符
strtok	抽取被指定字符分隔的子串
atoi	将字符串转换为整数(在cstdlib中声明)
atof	将字符串转换为实数(在cstdlib中声明)
itoa	将整数转换为字符串(在cstdlib中声明)

字符串操作库函数

●char * **strchr**(const char * str,int c);

寻找字符c在字符串str中第一次出现的位置。如果找到，就返回指向该位置的char*指针；如果str中不包含字符c,则返回NULL

●char * **strstr**(const char * str, const char * subStr);

寻找子串subStr在str中第一次出现的位置。如果找到，就返回指向该位置的指针；如果str不包含字符串subStr，则返回NULL

●int **stricmp**(const char * s1,const char * s2);

大小写无关的字符串比较。如果s1小于s2则返回负数；如果s1等于s2，返回0；s1大于s2,返回正数。不同编译器编译出来的程序，执行stricmp的结果就可能不同。

●int **strncmp**(const char * s1,const char * s2,int n);

比较s1前n个字符组成的子串和s2前n个字符组成的子串的大小。若长度不足n，则取整个串作为子串。返回值和strcmp类似。

●char * **strncpy**(char * dest, const char * src,int n);

拷贝src的前n个字符到dest。如果src长度大于或等于n，该函数不会自动往dest中写入‘\0’；若src长度不足n，则拷贝src的全部内容以及结尾的‘\0’到dest。

字符串操作库函数

●**char * strtok(char * str, const char * delim);**

连续调用该函数若干次，可以做到：从str中逐个抽取出被字符串delim中的字符分隔开的若干个子串。

●**int atoi(char *s);**

将字符串s里的内容转换成一个整型数返回。比如，如果字符串s的内容是“1234”，那么函数返回值就是1234。如果s格式不是一个整数，比如是"a12"，那么返回0。

●**double atof(char *s);**

将字符串s中的内容转换成实数返回。比如，"12.34"就会转换成12.34。如果s的格式不是一个实数，则返回0。

●**char * itoa(int value, char *string, int radix);**

将整型值value以radix进制表示法写入 string:

```
char szValue[20];
```

```
itoa( 27,szValue,10); //使得szValue的内容变为 "27"
```

```
itoa( 27,szValue,16); //使得szValue的内容变为"1b"
```

字符串操作库函数

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char s1[100] = "12345";
    char s2[100] = "abcdefg";
    char s3[100] = "ABCDE";
    strncat(s1,s2,3); // s1 = "12345abc"
    cout << "1) " << s1 << endl; //输出 1) 12345abc
    strncpy(s1,s3,3); // s3的前三个字符拷贝到s1,s1="ABC45abc"
    cout << "2) " << s1 << endl; //输出 2) ABC45abc
    strncpy(s2,s3,6); // s2 = "ABCDE"
    cout << "3) " << s2 << endl; //输出 3) ABCDE
    cout << "4) " << strncmp(s1,s3,3) << endl;
    //比较s1和s3的前三个字符, 比较结果是相等, 输出 4) 0
    char * p = strchr(s1,'B'); //在s1中查找 'B' 第一次出现的位置
```



```

if( p ) // 等价于 if( p!= NULL)
    cout << "5) " << p - s1 << ", " << *p << endl; //输出 5) 1,B
else
    cout << "5) Not Found" << endl;
p = strstr( s1, "45a"); //在s1中查找字符串 "45a"。 s1="ABC45abc"
if( p )
    cout << "6) " << p - s1 << ", " << p << endl; //输出 6) 3,45abc
else
    cout << "6) Not Found" << endl;
//以下演示strtok用法:
cout << "strtok usage demo:" << endl;
char str[] = "- This, a sample string, OK.";
//下面要从str逐个抽取出被" ,.-"这几个字符分隔的字符串
p = strtok (str, " ,.-"); //请注意, " ,.-"中的第一个字符是空格
while ( p != NULL) { //只要p不为NULL, 就说明找到了一个子串
    cout << p << endl;
    p = strtok(NULL, " ,.-"); //后续调用, 第一个参数必须是NULL
}
return 0;
}

```

This
a
sample
string
OK

void 指针

- void指针:

```
void * p;
```

- 可以用任何类型的指针对 void 指针进行赋值或初始化:

```
double d = 1.54;  
void * p = & d;  
void * p1;  
p1 = & d;
```

void 指针

- 因 `sizeof(void)` 没有定义，所以对于 `void *` 类型的指针`p`,

`*p` 无定义

`++p`, `--p`, `p += n`, `p+n`, `p-n` 等均无定义

内存操作库函数memset

头文件**cstring**中声明：

```
void * memset(void * dest,int ch,int n);
```

将从dest开始的**n个字节**，都设置成ch。返回值是dest。ch只有最低的字节起作用。

内存操作库函数memset

头文件cstring中声明：

```
void * memset(void * dest,int ch,int n);
```

将从dest开始的n个字节，都设置成ch。返回值是dest。ch只有最低的字节起作用。

例：将szName的前10个字符，都设置成'a'：

```
char szName[200] = "";  
memset( szName,'a',10);  
cout << szName << endl;  
=>aaaaaaaaaa
```

内存操作库函数memset

用memset函数将数组内容全部设置成0:

```
int a[100];  
memset(a,0,sizeof(a));
```

则数组a的每个元素都变成0

内存操作库函数memcpy

头文件**cstring**中声明：

```
void * memcpy(void * dest, void * src, int n);
```

将地址src开始的**n个字节**，拷贝到地址dest。返回值是dest。

内存操作库函数memcpy

头文件cstring中声明：

```
void * memcpy(void * dest, void * src, int n);
```

将地址src开始的n个字节，拷贝到地址dest。返回值是dest。

将数组a1的内容拷贝到数组a2中去，结果是a2[0] = a1[0], a2[1] = a1[1]……a2[9] = a1[9]：

```
int a1[10];  
int a2[10];  
memcpy( a2, a1, 10*sizeof(int));
```


如何编写内存操作库函数memcpy

```
void * MyMemcpy( void * dest , const void * src, int n)
{
    char * pDest = (char * )dest;
    char * pSrc = ( char * ) src;
    for( int i = 0; i < n; ++i ) {
        //逐个字节拷贝源块的内容到目的块
        * (pDest + i) = * ( pSrc + i );
    }
    return dest;
}
```

如何编写内存操作库函数memcpy

```
void * MyMemcpy( void * dest , const void * src, int n)
{
    char * pDest = (char * )dest;
    char * pSrc = ( char * ) src;
    for( int i = 0; i < n; ++i ) {
        //逐个字节拷贝源块的内容到目的块
        * (pDest + i) = * ( pSrc + i );
    }
    return dest;
}
```

有缺陷，在dest区间和src区间有重叠时可能出问题!!!



函数指针

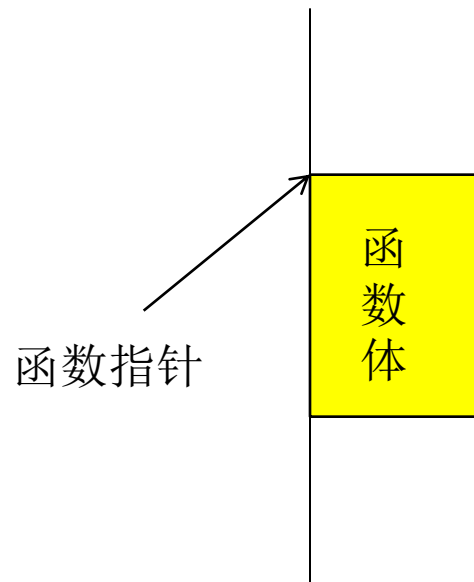
(教材P107)

基本概念

程序运行期间，每个函数都会占用一段连续的内存空间。而函数名就是该函数所占内存区域的起始地址(也称“入口地址”)。我们可以将函数的入口地址赋给一个指针变量，使该指针变量指向该函数。然后通过指针变量就可以调用这个函数。这种指向函数的指针变量称为“**函数指针**”。

基本概念

程序运行期间，每个函数都会占用一段连续的内存空间。而函数名就是该函数所占内存区域的起始地址(也称“入口地址”)。我们可以将函数的入口地址赋给一个指针变量，使该指针变量指向该函数。然后通过指针变量就可以调用这个函数。这种指向函数的指针变量称为“**函数指针**”。



定义形式

类型名 (* 指针变量名)(参数类型1, 参数类型2,...);

定义形式

类型名 (* 指针变量名)(参数类型1, 参数类型2,...);

例如:

```
int (*pf)(int ,char);
```

定义形式

类型名 (* 指针变量名)(参数类型1, 参数类型2,...);

例如:

```
int (*pf)(int ,char);
```

表示pf是一个函数指针，它所指向的函数，返回值类型应是int，该函数应有两个参数，第一个是int 类型，第二个是char类型

。

使用方法

可以用一个原型匹配的函数的名字给一个函数指针赋值。

要通过函数指针调用它所指向的函数，写法为：

函数指针名(实参表);

使用方法

```
#include <stdio.h>

void PrintMin(int a,int b) {
    if( a<b )
        printf("%d",a);
    else
        printf("%d",b);
}

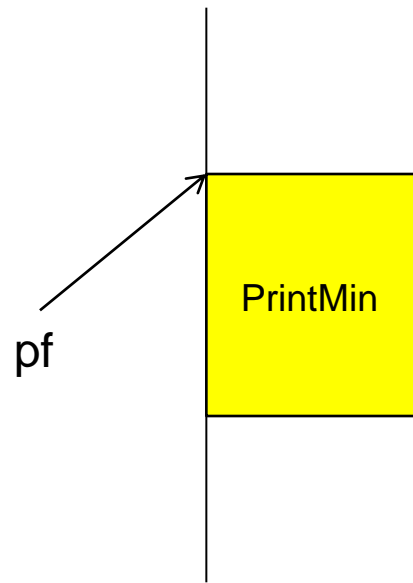
int main() {
    void (* pf)(int ,int);
    int x = 4, y = 5;
    pf = PrintMin;
    pf(x,y);
    return 0;
}
```

使用方法

```
#include <stdio.h>

void PrintMin(int a,int b) {
    if( a<b )
        printf("%d",a);
    else
        printf("%d",b);
}

int main() {
    void (* pf)(int ,int);
    int x = 4, y = 5;
    pf = PrintMin;
    pf(x,y);
    return 0;
}
```

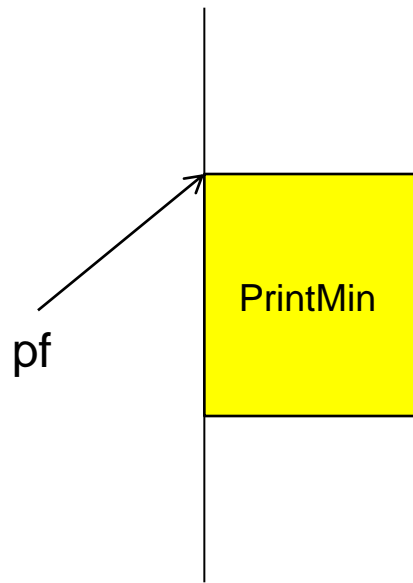


使用方法

```
#include <stdio.h>

void PrintMin(int a,int b) {
    if( a<b )
        printf("%d",a);
    else
        printf("%d",b);
}

int main() {
    void (* pf)(int ,int);
    int x = 4, y = 5;
    pf = PrintMin;
    pf(x,y);
    return 0;
}
```



输出结果：
4

函数指针和qsort库函数

C语言快速排序库函数：

```
void qsort(void *base, int nelem, unsigned int width,  
    int ( * pfCompare)( const void *, const void *));
```

可以对任意类型的数组进行排序

函数指针和qsort库函数

a[0]	a[1]	a[i]	a[n-1]
------	------	-------	------	-------	--------

对数组排序，需要知道：

函数指针和qsort库函数

a[0]	a[1]	a[i]	a[n-1]
------	------	-------	------	-------	--------

对数组排序，需要知道：

1) 数组起始地址

函数指针和qsort库函数

a[0]	a[1]	a[i]	a[n-1]
------	------	-------	------	-------	--------

对数组排序，需要知道：

- 1) 数组起始地址
- 2) 数组元素的个数

函数指针和qsort库函数

a[0]	a[1]	a[i]	a[n-1]
------	------	-------	------	-------	--------

对数组排序，需要知道：

- 1) 数组起始地址
- 2) 数组元素的个数
- 3) 每个元素的大小（由此可以算出每个元素的地址）

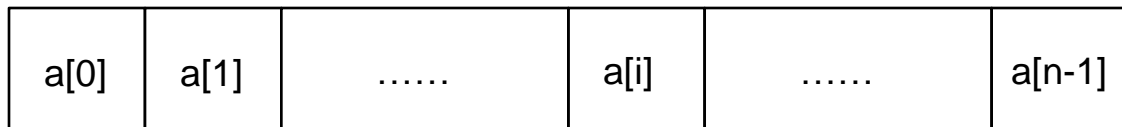
函数指针和qsort库函数

a[0]	a[1]	a[i]	a[n-1]
------	------	-------	------	-------	--------

对数组排序，需要知道：

- 1) 数组起始地址
- 2) 数组元素的个数
- 3) 每个元素的大小（由此可以算出每个元素的地址）
- 4) 元素谁在前谁在后的规则

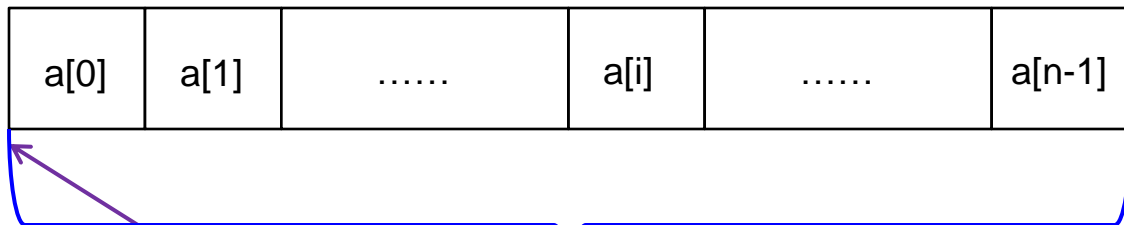
函数指针和qsort库函数



```
void qsort(void *base, int nelem, unsigned int width,  
int ( * pfCompare)( const void *, const void *));
```

base: 待排序数组的起始地址,

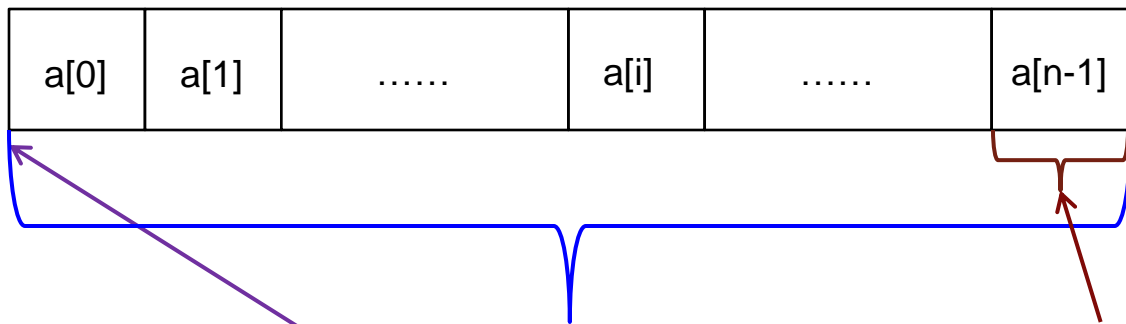
函数指针和qsort库函数



```
void qsort(void *base, int nelem, unsigned int width,  
int ( * pfCompare)( const void *, const void *));
```

base: 待排序数组的起始地址,
nelem: 待排序数组的元素个数,

函数指针和qsort库函数



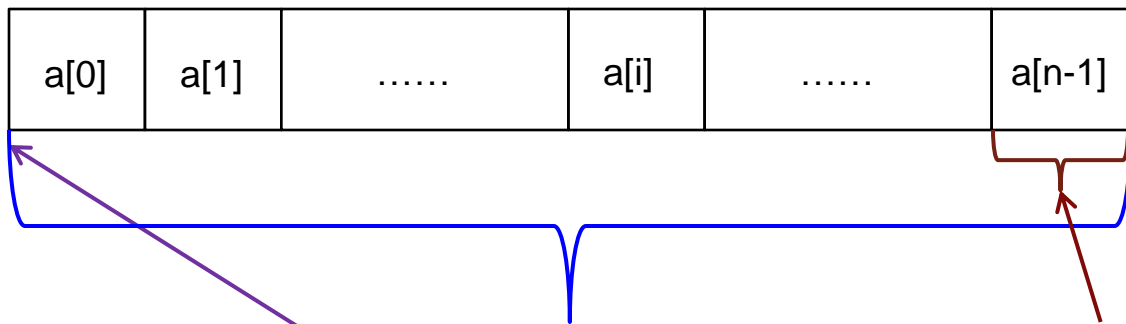
```
void qsort(void *base, int nelem, unsigned int width,  
int ( * pfCompare)( const void *, const void *));
```

base: 待排序数组的起始地址,

nelem: 待排序数组的元素个数,

width: 待排序数组的每个元素的大小 (以字节为单位)

函数指针和qsort库函数



```
void qsort(void *base, int nelem, unsigned int width,  
int (* pfCompare)( const void *, const void *));
```

base: 待排序数组的起始地址,

nelem: 待排序数组的元素个数,

width: 待排序数组的每个元素的大小 (以字节为单位)

pfCompare: 比较函数的地址

函数指针和qsort库函数

```
void qsort(void *base, int nelem, unsigned int width,  
    int ( * pfCompare)( const void *, const void *));
```

pfCompare: 函数指针，它指向一个“比较函数”。
该比较函数应为以下形式：

```
int 函数名(const void * elem1, const void * elem2);
```

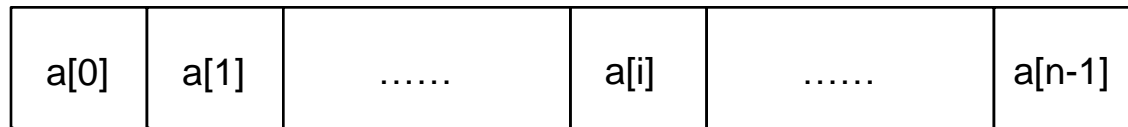
比较函数是程序员自己编写的

函数指针和qsort库函数


排序就是一个不断比较并交换位置的过程。

qsort函数在执行期间，会通过pfCompare指针调用“比较函数”，调用时将要比较的两个元素的地址传给“比较函数”，然后根据“比较函数”返回值判断两个元素哪个更应该排在前面。

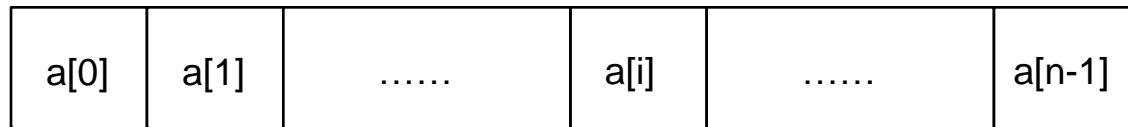
函数指针和qsort库函数



pfCompare(e1, e2);



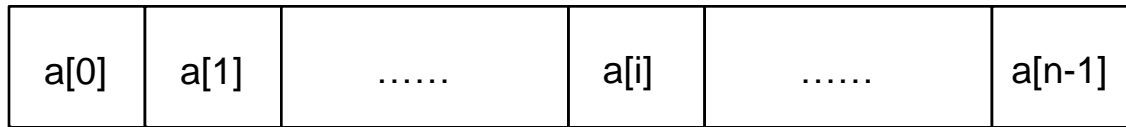
函数指针和qsort库函数



pfCompare(e1, e2);

int 比较函数名(const void * elem1, const void * elem2);

函数指针和qsort库函数



pfCompare(e1, e2);

int 比较函数名(const void * elem1, const void * elem2);

比较函数编写规则:

- 1) 如果 * elem1应该排在 * elem2前面, 则函数返回值是负整数
- 2) 如果 * elem1和* elem2哪个排在前面都行, 那么函数返回0
- 3) 如果 * elem1应该排在 * elem2后面, 则函数返回值是正整数

函数指针和qsort库函数

实例：

下面的程序，功能是调用qsort库函数，将一个unsigned int数组按照个数从小到大进行排序。比如 8，23，15三个数，按个数从小到大排序，就应该是 23，15，8

```
#include <stdio.h>
#include <stdlib.h>
int MyCompare( const void * elem1, const void * elem2 )
{
    unsigned int * p1, * p2;
    p1 = (unsigned int *) elem1; // “* elem1” 非法
    p2 = (unsigned int *) elem2; // “* elem2” 非法
    return (* p1 % 10) - (* p2 % 10 );
}
#define NUM 5
int main()
{
    unsigned int an[NUM] = { 8,123,11,10,4 };
    qsort( an,NUM,sizeof(unsigned int), MyCompare);
    for( int i = 0;i < NUM; i ++ )
        printf("%d ",an[i]);
    return 0;
}
```

输出结果:
10 11 123 4 8