



《计算概论A》课程 程序设计部分

指针(4) & 结构体

李 戈

北京大学 信息科学技术学院 软件研究所

2010年12月22日



北京大学



内容提要

■ 指针与函数

- ◆ 指针用做函数参数
- ◆ 指针用做函数返回值
- ◆ 指向函数的指针

■ 指针与结构体





指针做函数参数

■ 如下程序完成什么功能？

```
#include<iostream>
using namespace std;
void main( )
{
    int a[2] = {12, 5};
    cout<<"Max: "
        <<max(a, 2)
        <<endl;
}
```

```
int max(int *p, int n)
{
    int i = 0, temp = 0;
    for(i = 0; i < n; i++)
    {
        if(*(p+i) > temp)
        {
            temp = *(p+i);
        }
    }
    return temp;
}
```

指针做函数参数

■ 如下程序完成什么功能？

```
#include<iostream>
using namespace std;
void main( )
{
    int a[2] = {12, 5};
    cout<<"Max: "
        <<max(a, 2)
        <<endl;
}
```

```
int max(int *p, int n)
{
    int i = 0, temp = 0;
    for(i = 0; i < n; i++)
    {
        if(*(p+i) > temp)
        {
            temp = *(p+i);
        }
        else
        {
            *(p+i) = 0;
        }
    }
    return temp;
}
```

指针做函数参数

- 如下程序完成什么功能？

```
#include<iostream>
using namespace std;
void main( )
{
    int a[2] = {12, 5};
    cout<<"Max: "
        <<max(a, 2)
        <<endl;
}
```

```
int max(const int *p, int n)
{
    int i = 0, temp = 0;
    for(i = 0; i < n; i++)
    {
        if(*(p+i) > temp)
        {
            temp = *(p+i);
        }
        else
        {
            *(p+i) = 0; //错误
        }
    }
    return temp;
}
```

符号常量

■ 符号常量声明语句:

◆ 方式一: `const` 数据类型 常量名=常量值;

◆ 方式二: 数据类型 `const` 常量名=常量值;

```
#include <iostream.h>
```

```
void main()
```

```
{
```

```
    const float PI=3.14159f; // float const PI=3.14159f;
```

```
    float r;
```

```
    cout<<"请输入半径r: ";
```

```
    cin>>r;
```

```
    cout<<"圆面积为:"<<PI*r*r<<endl;
```

```
}
```



北京大學



指向符号常量的指针

■ 定义语句: `const int *p ;`

```
#include <iostream>
using namespace std;
void main()
{
    int a = 256;
    int *p = &a;
    *p = 257;
    cout<<*p<<endl;
}
```

```
#include <iostream>
using namespace std;
void main()
{
    int a = 256;
    const int *p = &a;
    *p = 257; //错误
    cout<<*p<<endl;
}
```





指向符号常量的指针

■ 用途

```
void mystrcpy(char *dest, const char *src)
```

```
{ .....
```

```
int main()
```

```
{
```

```
    char a[20] = "How are you!";
```

```
    char b[20];
```

```
    mystrcpy(b,a);
```

```
    cout<<b<<endl;
```

```
    return 0;
```

```
}
```

保证字符串src不被修改!



北京大学



指向符号常量的指针

■ 定义方式

```
int main()
{
    const int a = 78; const int b = 28; int c = 18;
    const int * pi = &a;
    *pi = 58;
    pi = &b; *pi = 68;

    pi = &c; *pi = 88;
    return 0;
}
```





指向符号常量的指针

■ 定义方式

```
int main()
```

```
{
```

```
    const int a = 78; const int b = 28; int c = 18;
```

```
    const int * pi = &a;
```

```
    *pi = 58;                //(error, *p不能被赋值)
```

```
    pi = &b; *pi = 68;
```

```
    //(error, 可以给pi重新赋值, 但 *p不能被赋值)
```

```
    pi = &c; *pi = 88;        (error, *p不能被赋值)
```

```
    return 0;
```

```
}
```





内容提要

■ 指针与函数

- ◆ 指针用做函数参数
- ◆ 指针用做函数返回值
- ◆ 指向函数的指针

■ 指针与结构体





返回指针值的函数

■ 函数的返回值可以是多种类型

◆ 返回整型数据的函数：

```
int max( int x, int y );
```

◆ 返回指针类型数据的函数

```
int *function( int x, int y );
```

- 函数名字前面表示函数的类型 “*”





返回指针值的函数

- 打印出第二行第三列的值

```
#include<iostream>
using namespace std;
void main(){
    int a[4][4]=
    {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
    12, 13, 14, 15, 16};
    int *p;
    p = get(a, 2, 3);
    cout<<*p<<endl;
}
```

```
int *get(int arr[ ][4],
          int n, int m)
{
    int *pt;
    pt = *(arr + n - 1) + m-1;
    return(pt);
}
```





返回指针值的函数

■ 判断程序的执行结果:

```
#include<iostream>
using namespace std;
int *getInt1()
{
    int value1 = 20;
    return &value1;
}
int main(){
    int *p;
    p = getInt1();
    cout << *p << endl;
    return 0;
}
```



返回指针值的函数

■ 判断程序的执行结果:

```
#include<iostream>
using namespace std;
int main(){
    int *p,*q;
    p = getInt1();
    q = getInt2();
    cout << *p << endl;
    return 0;
}
```

```
int *getInt1()
{
    int value1 = 20;
    return &value1;
}
int *getInt2()
{
    int value2 = 30;
    return &value2;
}
```




返回指针值的函数

■ 判断程序的执行结果:

```
#include<iostream>
using namespace std;
int main(){
    int *p,*q;
    p = getInt1();
    q = getInt2();
    cout << *p << endl;
    return 0;
}
```

```
int *getInt1()
{
    int value1 = 20;
    return &value1;
}
int *getInt2()
{
    int a[4][4]=
        {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
        11, 12, 13, 14, 15, 16};
    int value2 = 30;
    return &value2;
}
```



确保返回地址的意义

■ 返回一个处于生命周期中的变量的地址

◆ 返回全局变量的地址，而非局部变量的地址

```
#include<iostream.h>
int value1 = 20;
int value2 = 30;
int main()
{ int *p,*q;
  p = getInt1();
  q = getInt2();
  cout << *p << endl;
  return 0; }
```

```
int *getInt1()
{
    return &value1;
}

int *getInt2()
{
    return &value2;
}
```





确保返回的地址意义

■ 返回一个处于生命周期中的变量的地址

- ◆ 返回静态局部变量的地址，而非动态局部变量的地址

```
#include<iostream>
using namespace std;
int main(){
    int *p,*q;
    p = getInt1();
    q = getInt2();
    cout << *p << endl;
    return 0;
}
```

```
int *getInt1()
{
    static int value1 = 20;
    return &value1;
}
int *getInt2()
{
    auto int value2 = 30;
    return &value2;
}
```



什么是静态局部变量

■ 静态局部变量

- ◆ 有时希望函数中的局部变量的值在函数调用结束后不消失而保留原值，即其占用的存储单元不释放，在下一次该函数调用时，仍可以继续使用该变量；
- ◆ 用关键字static进行声明，可将变量指定为“静态局部变量”。

static int value1 = 20;



清华大学

```
#include<iostream.h>
void function( )
{
    auto int a = 0;
    static int b = 0;
    a = a+1;
    b = b+1;
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
}
void main()
{
    for(int i = 0; i < 5; i++)
    {
        function( );
        cout<<"Call Again!"<<endl;
    }
}
```

```
a = 1
b = 1
Call Again!
a = 1
b = 2
Call Again!
a = 1
b = 3
Call Again!
a = 1
b = 4
Call Again!
a = 1
b = 5
Call Again!
Press any key to continue
```



什么是静态局部变量

■ 动态局部变量

- ◆ 如果没有特别说明，一般定义的内部变量都是“动态局部变量”；
- ◆ 在调用该函数时系统会给它们分配存储空间，在函数调用结束时就自动释放这些存储空间。
- ◆ 可选用关键字“auto”

auto int value2 = 30;

auto不写则隐含确定为“自动存储类别”，它属于动态存储方式。



auto vs. static

```
#include<iostream.h>
```

```
int *getInt1()
```

```
{    static int value1 = 20;  
    return &value1; }
```

```
int *getInt2()
```

```
{    auto int value2 = 30;  
    return &value2; }
```

```
int main()
```

```
{  int *p,*q;  
    p = getInt1();  
    q = getInt2();  
    cout << *p << endl;  
    return 0; }
```

■ **动态变量**
value2的存
在范围

■ **静态变量**
value1存在
的范围



北京大學

确保返回的地址意义

■ 判断程序返回的结果:

```
#include<iostream>
using namespace std;
int main(){
    int *p,*q;
    p = getInt1();
    q = getInt2();
    cout << *p << endl;
    return 0;
}
```

```
int *getInt1()
{
    static int value1 = 20;
    return &value1;
}
int *getInt2()
{
    auto int value2 = 30;
    return &value2;
}
```





内容提要

■ 指针与函数

- ◆ 指针用做函数参数
- ◆ 指针用做函数返回值
- ◆ 指向函数的指针

■ 指针与结构体



北京大學



指向函数的指针

■ 已知

- ◆ C++程序中的一个函数，编译器进行编译时会给函数分配一个入口地址。
- ◆ 该地址被称为“函数的地址”或“函数的指针”。
- ◆ C++语言规定，函数的名字代表“函数的地址”。

■ 可以

- ◆ 定义一个指针变量pointer，并让该变量指向某个特定的函数；
- ◆ 指针变量的类型应与函数的类型相同；
 - 函数的类型 由它的返回值和参数表决定





指向函数的指针

```
void main( )
```

```
{  int max(int, int);    //用于声明的函数原型
```

```
    int (*p)(int, int);
```

```
        // 定义一个指向整型函数的指针变量;
```

```
int a, b, c;
```

```
p = max; //给 “整型函数型” 指针变量p赋值;
```

```
cin>>a>>b;
```

```
c = p(a, b); //利用指针变量p调用函数
```

```
cout<<"a= "<<a<<", b="<<b<<", max="<<c;
```

```
}
```





指向函数的指针

```
void main( )
```

```
{  int max(int, int);    //用于声明的函数原型
```

```
    int (*p)(int, int);
```

```
        // 定义一个指向整型函数的指针变量;
```

```
int a, b, c;
```

```
p = max; //给 “整型函数型” 指针变量p赋值;
```

```
cin>>a>>b;
```

```
c = (*p)(a, b);    //利用指针变量p调用函数
```

```
cout<<"a= "<<a<<", b="<<b<<", max="<<c;
```

```
}
```



指向函数的指针

■ `int (*p)(int, int)` 的意义是什么？

◆ 对比

- `int max (int, int)`

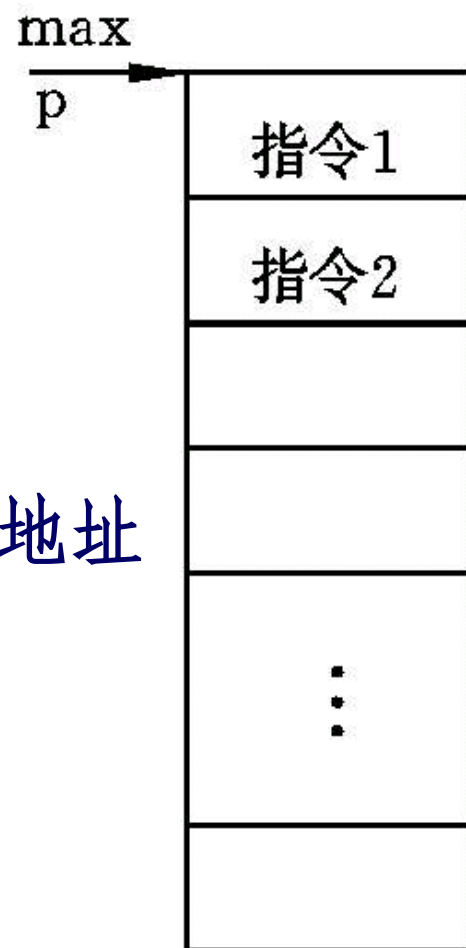
- `int (*p)(int, int)`

◆ 指针变量 `p` 的值是函数 `max` 的入口地址

■ 无意义的操作

◆ `p = max(a, b);`

◆ `p++, p--; ++p, --p;`



北京大学

指向函数的指针 的赋值

```
#include<iostream>
using namespace std;
int fn1(char x, char y) {return x+y;}
int fn2(int a)    {return a;}
int (*fp1)(char x, char y);
int (*fp2)(int s);
void main()
{ fp1 = fn1;      //ok
  fp2 = fn2;      //ok
  fp1 = fn2;      //error类型不一致
  fp2 = fp1;      //error类型不一致
  fp2 = fn2(5);   //error, fn2(5)不是函数的地址
  cout<<(*fp2)(5)<<endl;
  cout<<fp2(5)<<endl;
}
```





结构体



北京大学



试给出如下问题的解决方案

■ 问题

- ◆ 地震来袭，同学们纷纷解囊相助，向灾区捐款，班委会负责统计捐款的数目，要求：
- ◆ 依次录入同学们的姓名、学号、捐款数目；
- ◆ 要求按照统计捐款的数目，由大到小打印出姓名排名，捐款数目相同的同学，按照学号升序排列；

■ 请给出解决方案。



北京大学



什么是结构体

■ 问题:

- ◆ 现实世界中的事物都具有一些属性;
 - 例如, 学生有“学号”、“姓名”、“性别”、“年龄”等;
 - 如果在程序中分别定义“学号”..., 难记, 难用;
 - 难以体现出某些信息都是隶属于某个事物的;
- ◆ 在程序中, 希望能够用一个相对独立的数据结构来存储与某个事物相关的信息;
 - 能不能设计一种数据结构, 把这些分散的属性封装起来
 - 让他们“看起来”象一个整体, 用起来也可以作为整体来用

■ 结构体

- ◆ 是一种构造类型, 是由各种类型构造而成;
- ◆ 将各种不同类型但相关的数据“集合”起来;



北京大學



什么是结构体

- 声明一个名为“学生”的结构体

struct student \\结构体的名字为“**student**”;

{

int id; \\声明学号为**int**型;

char name[20]; \\声明姓名为字符数组;

char sex; \\声明性别为字符型;

int age; \\声明年龄为整型;

float score; \\声明成绩为实型;

char addr[30]; \\声明地址为字符数组

}; \\注意大括号后的“;”



北京大学



声明结构体类型的变量

■ 错误的理解

- ◆ “给出了**student**类型数据的定义，就可以使用**student**这个结构体了” **NO!!!!**

■ 声明的结构体是一种数据类型

- ◆ **student**仅仅是一种新生的 “数据类型”
- ◆ 从此，编译器认识一种 “**student**类型”，就像**int**型，**float**型，**char**型一样。

■ 必须利用所声明的结构体，定义 “结构体型的变量” 才能够使用

- ◆ 必须声明一个 “**student**类型” 的变量才能够使用
 - 就像不能够直接对 “**int**”， “**float**”... 进行计算操作





定义结构体类型的变量

■ 定义结构体变量的方式

(1) 直接用已声明的结构体类型定义变量名

student **student1, student2;**

(结构体类型名) (结构体变量名);

◆ 对比:

int a; (student 相当于 int)

float a; (student 相当于 float)



北京大學



定义结构体类型的变量

(2) 在声明类型的同时定义变量

```
struct student      \\结构体的名字为 “student”;  
{  
    int    id;      \\声明学号为int型;  
    char   name[20]; \\声明姓名为字符数组;  
    char   sex;      \\声明性别为字符型;  
    int    age;      \\声明年龄为整型;  
    float  score;    \\声明成绩为实型;  
    char   addr[30];  \\声明地址为字符数组  
} lige_1, lige_2;    \\注意最后的 “;”
```





定义结构体类型的变量

(3) 直接定义结构体变量

```
struct                                \\声明无名字结构体;  
{   int    id;                       \\声明学号为int型;  
    char   name[20]; \\声明姓名为字符数组;  
    char   sex;                \\声明性别为字符型;  
    int    age;                \\声明年龄为整型;  
    float  score;              \\声明成绩为实型;  
    char   addr[30]; \\声明地址为字符数组  
} lige_1, lige_2;                \\注意最后的 “;”
```





结构体可以嵌套

```
struct student
{ int    id;
  char   name[20];
  char   sex;
  int    age;
  date birthday;
  char   addr[30];
} student1, student2;
```

```
struct date
{
  int    month;
  int    day;
  int    year;
};
```





结构体变量的引用

- 引用结构体变量中成员的方式为

结构体变量名.成员名

- ◆ 如: `student1.id=10010;`

- `student1.birthday.month = 10;`

- 不能将一个结构体变量作为一个整体进行输入和输出

- ◆ 不正确的引用: `cout<<student1; cin>>student1;`

- 只能对结构体变量中的各个成员分别进行输入和输出

- ◆ 正确的引用: `cin>>student1.id; cout<<student1.id;`





结构体变量的引用

■ 结构体中的成员，可以单独使用，相当于普通变量：

◆ `student1.age = student2.age;`

◆ `student1.age++;`

■ 成员名可以与程序中的变量名相同：

```
struct date
{
    int    month;
    int    day;
    int    year;
};
```

```
main( )
{
    int day = 7;
    struct date Christmas = {12, 25, 2007};
    cout<<day<<endl;
    cout<<Christmas.day<<endl;
}
```





结构体变量的初始化

■ 结构体可以在定义时进行初始化

```
struct date
{ int month;
  int day;
  int year; };
```

```
struct student
{ int num;
```

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

```
  char sex;
```

```
  int age;
```

```
  struct date birthday;
```

```
  char addr[30];
```

```
}student1={121, "zhang", 'M', 20, {12, 30, 2000}, "PKU"},
```

```
student2={122, "wang", 'M', 20, {12, 30, 2000}, "PKU"};
```

```
student student3
```

```
    = {123, "zhao", 'M', 20, {12, 30, 2000}, "PKU"};
```



北京大学



结构体变量的赋值

- 类型相同的结构体变量可以进行赋值

- ◆ 例如:

要将student1和student2互换

```
temp = student1;
```

```
student1 = student2;
```

```
student2 = temp;
```

- ◆ (temp也必须是相同类型结构体变量)



北京大学



结构体变量的赋值

- 不同类型的结构体变量，即使成员完全一样也不允许互相赋值。

◆ 如下例中若： `person1 = student1;`，则编译错误！

```
struct student
{ int  stuNo;
  int  name[20];
  char sex;
  int  age;
  char addr[30];
}student1, student2;
```

```
struct person
{ int  stuNo;
  int  name[20];
  char sex;
  int  age;
  char addr[30];
}person1, person2;
```





结构体数组

■ 结构体数组

- ◆ 每个数组元素都是一个结构体变量

■ 举例

- ◆ `student stu[3];`

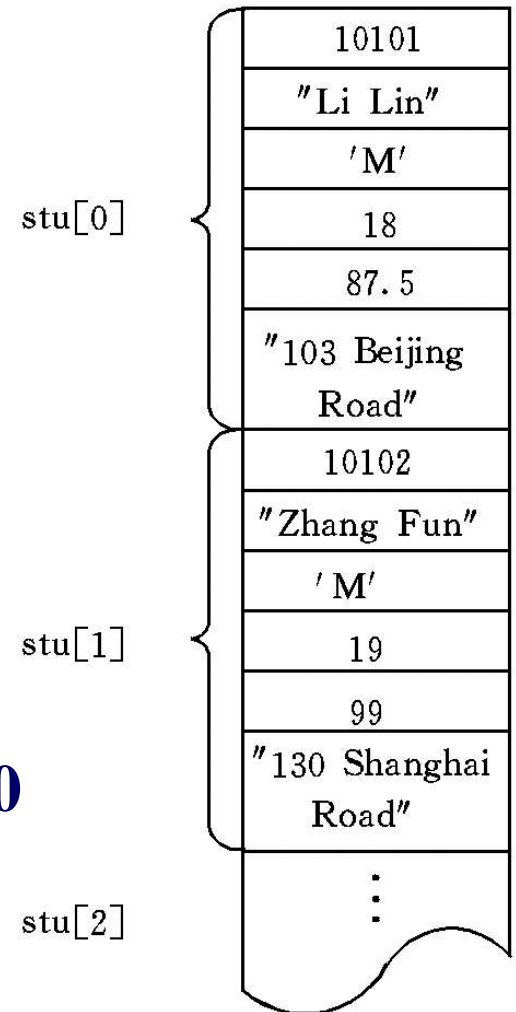
	num	name	sex	age	score	addr
stu[0]	10101	Li Lin	M	18	87.5	103 Beijing Road
stu[1]	10102	Zhang Fun	M	19	99	130 Shanghai Road
stu[2]	10104	Wang Min	F	20	78.5	1010 Zhongshan Road



结构体数组的初始化

```
struct student
{   int    num;
    char   name[20];
    char   sex;
    int    age;
    float  score;
    char   add[30];
```

```
} stu[3] = {
    { 10101, "Li Lin", 'M', 18, 87.5, "103
      Beijing Road" },
    { 10102, "Zhang Fun", 'M', 19, 99, "130
      Shanghai Road" },
    { 10104, "Wang Min", 'F', 20, 78.5, "1010
      Zhongshan Road" }
};
```



结构体数组的引用

```
struct date
{   int   month;
    int   day;
    int   year; };
struct student
{   int   num;
    char   name[20];
    char   sex;
    int    age;
    struct date birthday;
    char   addr[30];
}mystudents={
    {121, "zhang", 'M', 20, {12, 30, 2000}, "PKU"},
    {122, "wang", 'M', 20, {12, 30, 2000}, "PKU"},
    {123, "zhao", 'M', 20, {12, 30, 2000}, "PKU"}
};
```

■ 正确的引用：
mystudents[0].age;
mystudents[0].date.day;



指向结构体数组的指针

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
};

struct student stu[3]=
{
    {10101, "Li Lin",'M',18},
    {10102, "Zhang
Fun",'M',19},
    {10104, "Wang
Min",'F',20}
};
```

```
void main( )
{
    struct student*p;
    cout<<"No."<<"Name"<<"sex"<<"age";
    for(p = stu; p<stu+3; p++)
        cout<<p->num<< p->name<<p->sex<<
        p->age;
}
```

运行结果如下:

No.	Name	sex	age
10101	Li Lin	M	18
10102	Zhang Fun	M	19
10104	Wang Min	F	20



北京大学



指向运算符

- 在C++语言中，为了方便和直观，可以把`(*p).num`改用`p→num`来代替，它表示`*p`所指向的结构体变量中的`num`成员。

◆ 以下三种形式等价：

① 结构体变量. 成员名

② `(*p).成员名`

③ `p->成员名`

- 其中`->`称为指向运算符。



北京大学

结构数组应用示例(1)

- 对候选人的得票进行统计

```
void main( )
```

```
{  char name[5][20]={"A","B","C","D","E"};
```

```
    int count[5]={0,0,0,0,0};
```

```
    char vote[20];
```

```
    for (int i=1; i<=10; i++)
```

```
    {    cin.getline(vote,20);
```

```
        for (int j=0; j<5; ++)
```

```
            if (strcmp(vote, name[j])==0)
```

```
                count[j]++;
```

```
    }
```

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

```
    {    cout<<setw(5)<<name[i]<<setw(2)<<count[i]<<endl;    }
```

```
}
```



北京大学



结构数组应用示例(1)

```
void main()
{  struct person
   {   char name[20];
       int count;
   } candidate[5] = {"A", 0, "B", 0, "C", 0, "D", 0, "E", 0};
   char vote[20];
   for (int i=1; i<=10; i++)
   {   cin.getline(vote,20);
       for(int j=0; j<5; j++)
           if (strcmp(vote, candidate[j].name)==0)
               candidate[j].count++;
   }
   for (i=0;i<3;i++)
       cout<<setw(5)<<candidate[i].name
           <<setw(2)<<candidate[i].count<<endl;
}
```

```
#include<iostream.h>
#include<string.h>
#include <iomanip.h>
void main()
{  char ch; int k;
   struct alpha
   {   char name;
       int count;
   }letter[26], t;
   for(int i = 0; i < 26; i++) //对结构体变量letter进行初始化
   {   letter[i].name = 'a' + i;
       letter[i].count = 0;
   }
   for (i = 0; i < 50; i++) //输入字母并记录每个字母的出现次数
   {   cin >> ch;
       k = ch - 'a';
       letter[k].count++;
   }
}
```

结构数组应用示例 (2)

- 从键盘上输入50个字母（小写），按字母出现的频数由大到小排序

//定义结构体用于存放字母及出现次数

//输入字母并记录每个字母的出现次数



结构数组应用示例 (2)

- 从键盘上输入50个字母（小写），按字母出现的频数由大到小排序

```
for (i = 0; i < 25; i++)    //排序运算
    for (int j = 0; j < 25 - i; j++)
        if (letter[j].count < letter[j+1].count)
        {
            t = letter[j];
            letter[j] = letter[j + 1];
            letter[j + 1] = t;
        }
for (i=0; i<25; i++)
    cout<<letter[i].name<<letter[i].count;
}
```





好好想想，有没有问题？

谢谢！



清华大学