

计算机程序设计基础(1)

--- C语言程序设计(11A)

孙甲松

sunjiasong@tsinghua.edu.cn

电子工程系 信息认知与智能系统研究所

罗姆楼6-104

电话: 13901216180/62796193

2022.11.

第11章 结构体与联合体

11.1 结构体类型变量

11.1.1 结构体类型变量的定义与引用

11.1.2 结构体的嵌套

11.1.3 结构体类型变量的初始化

11.1.4 结构体与函数

11.2 结构体数组

11.2.1 结构体类型数组的定义与引用

11.2.2 结构体类型数组作为函数参数

11.3 结构体与指针

11.3.1 结构体类型指针变量的定义
与引用

11.3.2 结构体类型指针作为函数参数

11.3.3 结构体的大小与
#pragma中pack的关系

11.4 链表

11.4.1 链表的基本概念

11.4.2 链表的基本运算

11.4.3 多项式的表示与运算

11.5 联合体

11.6 枚举类型与自定义类型名

11.6.1 枚举类型

11.6.2 自定义类型名

11.1 结构体类型变量

11.1.1 结构体类型变量的定义与引用

1. 结构体类型变量的定义

C编译系统定义了基本数据类型，用户可以用系统所提供的类型名(如float, double, char, short, int, long, long long等)直接定义相应类型的变量。

数组是相同数据类型的集合，如果需要定义基本数据类型的数组，也只需要用这些基本数据类型名直接定义。

结构体类型是一种复合类型，各种不同的结构体中的成员个数以及各成员的数据类型可能是各不相同的。具体的结构体类型由用户根据实际需要自己定义，C编译系统提供了用户自己定义结构体类型的机制，以使用所定义的复合类型来描述现实世界中的复杂对象。

● 定义结构体类型

形式:

```
struct 结构体类型名  
{ 成员表 };
```

其中 **struct** 是用于定义具体结构体类型的关键字, 在“成员表”中可以定义该类型中有哪些成员, 各成员属于什么数据类型。

例如, 定义“日期”的结构体类型如下:

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

在说明结构体类型中各成员的时候, 下列定义:

```
struct date
```

```
{ int year, month, day ; };
```

也是正确的!

但通常在定义结构体类型时, 其中的各成员分别说明, 这样便于阅读与注释。

-
- 另外还要强调的是, 结构体类型名 (如本定义中的 **date**) 定义的是类型名, 而不是变量名, 就好象整型的类型名为 **int**, 双精度实型的类型名为 **double**, 字符型的类型名为 **char** 一样, 只不过整型、双精度实型、字符型等基本数据类型是C编译系统已经定义好的, 用户可以直接用它们来定义相应类型的变量, 而结构体类型是用户根据数据处理的需要临时定义的一种类型。

例如, 用**struct date**类型定义变量**a**, 并同时赋初值。

```
#include <stdio.h>
```

```
struct date
```

```
{ int year;
```

```
    int month;
```

```
    int day;
```

```
}; /* 注意: 最后这个分号不能少! */
```

```
main( )
```

```
{ struct date a = {2022, 11, 25}; /* 用struct date类型定义变量a */
```

```
    printf("%02d/%02d/%d\n", a.month, a.day, a.year);
```

```
}
```

编译链接没有错误, 运行结果:

11/25/2022

请按任意键继续...

● 定义结构体类型变量

将定义结构体类型与定义结构体类型变量分开说明



● **struct** 结构体类型名 变量表;

例如,当前面定义了结构体**date**类型 (“日期”类型)后,就可以定义该类型的变量如下:

```
struct date birthday, x, y;
```

● 需要说明的是,在定义结构体类型变量时,在定义结构体类型变量时不能只使用结构体类型名,而应该使用结构体类型的全称。**struct** 关键字不能省略。但C++中可以省略**struct**。

例如,下列定义是错误的:

```
date birthday, x, y;  
student a, b, st;
```

下列定义是正确的:

```
struct date birthday, x, y;  
struct student a, b, st;
```

同时定义结构体类型和变量



- **struct** 结构体类型名
{ 成员表 } 变量表;

例如,

```
struct date
{ int year;
  int month;
  int day;
} birthday, x, y;
struct date z, abc;
```

- 在定义结构体类型的同时又定义了结构体类型变量后, 并不意味着在程序中就不能再定义此结构体类型的变量了, 如果需要, 在程序中还可以继续定义此结构体类型的其他变量。

- 无名结构体类型：直接定义结构体类型变量而没有类型名



struct

{ 成员表 } 变量表;

例如:

```
struct
{
    int num;
    char name[10];
    char gender;
    int age;
    float score;
} a, b, st;
```

- 需要指出的是, 如果在函数体外定义了一个结构体类型, 则从定义位置开始到整个程序文件结束之前的所有函数中均可用该结构体类型定义该类型的变量。文件作用域
- 但在函数体内所定义的结构体类型, 只能在该函数体内能用来定义该类型的变量。函数作用域
- 结构体类型定义的作用域与普通变量定义的作用域是相同的。但结构体类型定义没有全局作用域。同一个结构体类型只能在各个文件中都重新定义。

2. 结构体类型变量的引用

方式

结构体变量名.成员名

- 其中“.”为**成员运算符**, 它的执行优先级是C语言中最高的。
- 结构体变量中的每个成员与普通变量一样, 可以进行各种运算。

```
struct student
{
    int num;
    char name[10];
    char gender;
    int age;
    float score;
} st;
st.num=115;
st.name[0]='M'; st.name[1]='a'; st.name[2]='\0';
/* 或 strcpy(st.name, "Ma"); */
st.gender='M';
st.age=19;
st.score=95.0;
scanf("%d", &st.num);
printf("%s", st.name);
```

● 特别要指出的是, 如果结构体变量中的某成员是一个数组, 则在为该成员赋值时, 与普通数组一样, 必须对该成员中的元素逐个赋值。

11.1.2 结构体的嵌套

结构体类型的定义可以嵌套

定义一个结构体“时间”
类型 **time** 如下:

```
struct time
{ int hour;    /* 时 */
  int minute; /* 分 */
  int second; /* 秒 */
};
```

定义一个结构体“日期”
类型 **date**:

```
struct date
{ int year;    /* 年 */
  int month;   /* 月 */
  int day;     /* 日 */
  struct time t; /* 结构体“时间”成员 */
};
```

在这个定义中, 结构体“日期”类型 **date** 中有一个成员又属于前面定义的结构体“时间”类型 **time**。这就是结构体的嵌套。

如果定义了结构体**date**类型（即**date**）变量**d**：

```
struct date d;
```

则下面是其引用结构体**date**类型变量**d**中各成员的例子：

```
d.year    /* 结构体date类型变量d的成员year (年) */
```

```
d.month   /* 结构体date类型变量d的成员month (月) */
```

```
d.day     /* 结构体date类型变量d的成员day (日) */
```

```
d.t.hour  /* 结构体date类型变量d的成员t (结构体time类型)
           中的成员hour (时) */
```

```
d.t.minute /* 结构体date类型变量d的成员t (结构体time类型)
           中的成员minute (分) */
```

```
d.t.second /* 结构体date类型变量d的成员t (结构体time类型)
           中的成员second (秒) */
```

“.” 运算符的运算顺序**自左至右**

11.1.3 结构体类型变量的初始化

与普通变量一样, 在定义结构体类型变量的同时也可以对结构体类型变量赋初值。其原理与普通数组的初始化一样。

【例11-1】 设有下列C程序:

```
#include <stdio.h>
struct student
{ int num;
  char name[10];
  char gender;
  int age;
  float score;
};
struct time
{ int hour;
  int minute;
  int second;
};
```

```
struct date
{ int year;
  int month;
  int day;
  struct time t;
};
main( )
{ struct student st={101, "Zhang", 'M', 19, 89.0};
  struct date xy={2022, 11, 25, {10, 34, 55}};
  printf("st=%6d%8s%3c%4d%7.2f\n",
        st.num, st.name, st.gender, st.age, st.score);
  printf("date=%d/%02d/%02d/%d:%d:%d\n", xy.year,
        xy.month, xy.day, xy.t.hour, xy.t.minute, xy.t.second);
}
```

程序的运行结果为:

```
st= 101  Zhang M 19 89.00
date=2022/11/25/10:34:55
```

- 由于一个结构体类型变量往往包括多个成员, 因此, 在为结构体类型变量初始化时, 要用一对花括号将所有的成员数据括起来。

```
struct student st={101, "Zhang", 'M', 19, 89.0};
```

- 如果是结构体的嵌套, 则对内层结构体中的所有成员数据也可以用一对花括号括起来, 但也可以不用花括号括起来。

```
struct date xy={2022, 11, 25, {10, 34, 55}}; 与  
struct date xy={2022, 11, 25, 10, 34, 55}; 是等价的。
```

- 如果在结构体中有数组成员, 则对成员数组中的元素可以另用一对花括号括起来, 也可以不用花括号括起来。
- 但需要注意的是, 如果上述结构体类型中的数组成员不是最后一个成员, 且不是对该数组成员中的所有元素赋初值, 则对成员数组中的元素必须另用一对花括号括起来。

```
struct student
{ int num;
  char name[10];
  char gender;
  int age;
  float score[3];
};
```

则下列两个初始化该结构体类型变量的语句是等价的：

```
struct student st={101, "Zhang", 'M', 19, {89.0, 93.0}};
```

```
struct student st={101, "Zhang", 'M', 19, 89.0, 93.0};
```

其中成员st.score[2]的初值默认为0.0。

```
struct student
{ int num;
  float score[3];
  char name[10];
  char gender;
  int age;
};
```

则下列初始化语句是合法的：

```
struct student st={101,{89.0,93.0},"Zhang", 'M',19};
```

其中成员st.score[2]的初值默认为0.0。

但下列初始化语句是错误的：

```
struct student st={101, 89.0, 93.0, "Zhang", 'M',19};
```

成员st.score[2]将被赋值为"Zhang", 类型不一致，会导致编译错误。


11.1.4 结构体与函数

● 结构体类型变量的成员作为函数参数


在结构体类型变量中的成员作为函数参数的情况下，被调用函数中的形参是一般变量，而调用函数中的实参是结构体类型变量中的一个成员，但要求它们的类型应一致。

【例11-2】设有下列C程序：

```
#include <stdio.h>
struct student
{ int num;
  char name[10];
  char gender;
  int age;
  float score;
};
```



```
void change(float t)
{ printf("score=%7.2f\n",t);
  t=95.0;
  printf("score=%7.2f\n",t);
}
```



```
main()  
{ struct student st={101,"Zhang",'M',19,89.0};  
  printf("st=%6d%8s%3c%4d%7.2f\n",  
        st.num, st.name, st.gender, st.age, st.score);  
  change(st.score);  
  printf("st=%6d%8s%3c%4d%7.2f\n",  
        st.num, st.name, st.gender, st.age, st.score);  
}
```

程序运行结果为

```
st= 101  Zhang M 19 89.00  
score= 89.00  
score= 95.00  
st= 101  Zhang M 19 89.00
```


若结构体成员做实参，函数不能改变结构体成员的值。因为实参结构体成员st.score是以值传递方式复制到形参t上，t的改变与st.score无关！

● 结构体类型变量作为函数参数


在结构体类型的变量作为函数参数的情况下，被调用函数中的形参是结构体类型的变量，调用函数中的实参也是结构体类型的变量，但要求它们属于同一个结构体类型。

【例11-3】 设有下列C程序：

```
#include <stdio.h>
#include <string.h>
struct student
{ int num;
  char name[10];
  char gender;
  int age;
  float score;
};
```



```
void change(struct student t)
{ printf("t=%6d%8s%3c%4d%7.2f\n",
        t.num, t.name, t.gender, t.age, t.score);
  t.score=95.0; strcpy(t.name, "Huang");
  printf("t=%6d%8s%3c%4d%7.2f\n",
        t.num, t.name, t.gender, t.age, t.score);
}
```



```
main( )
{ struct student st={101, "Zhang", 'M', 19, 89.0};
  printf("st=%6d%8s%3c%4d%7.2f\n",
        st.num, st.name, st.gender, st.age, st.score);
  change(st);
  printf("st=%6d%8s%3c%4d%7.2f\n",
        st.num, st.name, st.gender, st.age, st.score);
}
```

程序运行结果为:

st=	101	Zhang	M	19	89.00
t=	101	Zhang	M	19	89.00
t=	101	Huang	M	19	95.00
st=	101	Zhang	M	19	89.00

若结构体变量做实参，函数不能改变结构体变量成员的值(包括数组)。因为实参结构体变量`st`是以值传递方式复制到形参`t`上（如果结构体成员是数组，则整个数组也会被复制），形参`t`的改变与实参`st`无关！

● 返回值为结构体类型的函数

与定义标准数据类型函数一样, C语言也允许定义返回值为结构体类型的函数。结构体类型函数的返回值是结构体类型的数据。

【例11-4】设有下列C程序:

```
#include <stdio.h>
struct date
{ int year;
  int month;
  int day;
};
struct date f( )
{ struct date t={2022, 11, 25};
  return t;
}
```

```
main( )
{ struct date xy;
  xy=f( );
  printf("date=%d/%02d/%02d\n",
        xy.year, xy.month, xy.day);
}
```

程序的运行结果为: date=2022/11/25

11.2 结构体数组

11.2.1 结构体类型数组的定义与引用

- 与整型数组、实型数组、字符型数组一样，在程序中也可以定义结构体类型的数组，但同一个结构体数组中的元素应为同一种结构体类型。

例如：

```
struct student
{
    int num;
    char name[10];
    char gender;
    int age;
    float score[3];
} stu[10];
```

num 学号	name 姓名	gender 性别	age 年龄	score[0] 成绩1	score[1] 成绩2	score[2] 成绩3

定义了student类型的一个数组stu，可存放10个学生的情况。每一个学生的情况包括：学号(num)、姓名(name[10])、性别(gender)、年龄(age)、三门课成绩(score[3])。实际上，定义了该结构体数组后，相当于开辟了一个如上表所示的表格空间。

【例11-5】给定学生成绩登记表如表11-2所示。

表 11.2 学生成绩登记表

学 号	姓 名	性 别	年 龄	成绩 1	成绩 2	平均成绩
101	Zhang	M	19	95.5	64.0	
102	Wang	F	18	92.0	97.0	
103	Zhao	M	19	85.0	78.0	
104	Li	M	20	96.0	88.0	
105	Gou	M	19	91.0	96.0	
106	Lin	M	18	93.0	78.0	
107	Ma	F	18	98.0	97.0	
108	Zhen	M	21	89.0	93.0	
109	Xu	M	19	88.0	90.0	
110	Mao	F	18	94.0	90.0	

【例11-5】利用结构体数组计算表11-2中给定的两门课程成绩的平均成绩,最后输出该学生成绩登记表。

```
#define STU struct \
student
STU
{   int num;
    char name[10];
    char gender;
    int age;
    float score[3];
};
```

```
#include <stdio.h>
main( )
{   int k;
    STU stu[10]= /*结构体数组初始化*/
    {{101, "Zhang", 'M', 19, 95.0, 64.0 },
     {102, "Wang", 'F', 18, 92.0, 97.0 },
     {103, "Zhao", 'M', 19, 85.0, 78.0 },
     {104, "Li",   'M', 20, 96.0, 88.0 },
     {105, "Gou",  'M', 19, 91.0, 96.0 },
     {106, "Lin",  'M', 18, 93.0, 78.0 },
     {107, "Ma",   'F', 18, 98.0, 97.0 },
     {108, "Zhen", 'M', 21, 89.0, 93.0 },
     {109, "Xu",   'M', 19, 88.0, 90.0 },
     {110, "Mao",  'F', 18, 94.0, 90.0 }
    };
};
```

score[0]表示成绩1, score[1]表示成绩2, score[2]表示平均成绩。


```
for (k=0; k<=9; k++) /*计算平均成绩,并输出每个学生的全部信息*/  
{ stu[k].score[2]=(stu[k].score[0] + stu[k].score[1])/2;  
  printf("%-8d%-10s%-5c%-6d%-7.2f%-7.2f%-7.2f\n",  
        stu[k].num, stu[k].name, stu[k].gender, stu[k].age,  
        stu[k].score[0], stu[k].score[1], stu[k].score[2]);  
}  
}
```

在输出格式说明符中的“-”号表示输出的各项左对齐，即在输出项目的实际位数小于宽度说明时，右边用空格补满。

● 其运行结果为:

101	Zhang	M	19	95.00	64.00	79.50
102	Wang	F	18	92.00	97.00	94.50
103	Zhao	M	19	85.00	78.00	81.50
104	Li	M	20	96.00	88.00	92.00
105	Gou	M	19	91.00	96.00	93.50
106	Lin	M	18	93.00	78.00	85.50
107	Ma	F	18	98.00	97.00	97.50
108	Zhen	M	21	89.00	93.00	91.00
109	Xu	M	19	88.00	90.00	89.00
110	Mao	F	18	94.00	90.00	92.00

11.2.2 结构体类型数组作为函数参数

● 结构体类型数组也能作为函数参数，并且形参与实参结合的方式与基本数据类型的数组完全一样。因为结构体类型形参数组与结构体类型实参数组是同一个存储空间。**传数组首地址。**

【例11-6】

```
#define STU struct student
STU
{   int num;
    char name[10];
    char gender;
    int age;
    float score[3];
};
```

```
void p(STU t[ ], int n)
{   int k;
    for (k=0; k < n; k++)
        t[k].score[2]=(t[k].score[0]+
                        t[k].score[1])/2;
}
```

p函数循环计算每一个人的两门课的平均成绩。

```
#include <stdio.h>
main( )
{ int k;
  STU stu[10]=
    { { 101, "Zhang", 'M', 19, 95.0, 64.0 },
      { 102, "Wang", 'F', 18, 92.0, 97.0 },
      { 103, "Zhao", 'M', 19, 85.0, 78.0 },
      { 104, "Li", 'M', 20, 96.0, 88.0 },
      { 105, "Gou", 'M', 19, 91.0, 96.0 },
      { 106, "Lin", 'M', 18, 93.0, 78.0 },
      { 107, "Ma", 'F', 18, 98.0, 97.0 },
      { 108, "Zhen", 'M', 21, 89.0, 93.0 },
      { 109, "Xu", 'M', 19, 88.0, 90.0 },
      { 110, "Mao", 'F', 18, 94.0, 90.0 }
    }; /* score[2] 未赋初值,自动赋值为0.0 */
```

```
for (k=0; k<=9; k++)
    printf("%-8d%-10s%-5c%-6d%-7.2f%-7.2f%-7.2f\n",
           stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
           stu[k].score[0], stu[k].score[1], stu[k].score[2]);
printf("—————\n");
p(stu,10); /* 调用p函数循环计算每一个人的平均成绩 */
for (k=0; k<=9; k++)
    printf("%-8d%-10s%-5c%-6d%-7.2f%-7.2f%-7.2f\n",
           stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
           stu[k].score[0], stu[k].score[1], stu[k].score[2]);
}
```

程序运行结果如下;

101	Zhang	M	19	95.00	64.00	0.00
102	Wang	F	18	92.00	97.00	0.00
103	Zhao	M	19	85.00	78.00	0.00
104	Li	M	20	96.00	88.00	0.00
105	Gou	M	19	91.00	96.00	0.00
106	Lin	M	18	93.00	78.00	0.00
107	Ma	F	18	98.00	97.00	0.00
108	Zhen	M	21	89.00	93.00	0.00
109	Xu	M	19	88.00	90.00	0.00
110	Mao	F	18	94.00	90.00	0.00

其中前半部分是计算平均成绩前的学生情况表,

后半部分是计算平均成绩后的学生情况表。

101	Zhang	M	19	95.00	64.00	79.50
102	Wang	F	18	92.00	97.00	94.50
103	Zhao	M	19	85.00	78.00	81.50
104	Li	M	20	96.00	88.00	92.00
105	Gou	M	19	91.00	96.00	93.50
106	Lin	M	18	93.00	78.00	85.50
107	Ma	F	18	98.00	97.00	97.50
108	Zhen	M	21	89.00	93.00	91.00
109	Xu	M	19	88.00	90.00	89.00
110	Mao	F	18	94.00	90.00	92.00

C:\WINDOWS\system32\cmd.exe

101	Zhang	M	19	95.00	64.00	0.00
102	Wang	F	18	92.00	97.00	0.00
103	Zhao	M	19	85.00	78.00	0.00
104	Li	M	20	96.00	88.00	0.00
105	Gou	M	19	91.00	96.00	0.00
106	Lin	M	18	93.00	78.00	0.00
107	Ma	F	18	98.00	97.00	0.00
108	Zhen	M	21	89.00	93.00	0.00
109	Xu	M	19	88.00	90.00	0.00
110	Mao	F	18	94.00	90.00	0.00

101	Zhang	M	19	95.00	64.00	79.50
102	Wang	F	18	92.00	97.00	94.50
103	Zhao	M	19	85.00	78.00	81.50
104	Li	M	20	96.00	88.00	92.00
105	Gou	M	19	91.00	96.00	93.50
106	Lin	M	18	93.00	78.00	85.50
107	Ma	F	18	98.00	97.00	97.50
108	Zhen	M	21	89.00	93.00	91.00
109	Xu	M	19	88.00	90.00	89.00
110	Mao	F	18	94.00	90.00	92.00

请按任意键继续. . .

11.3 结构体与指针

11.3.1 结构体类型指针变量的定义与引用

- 结构体类型的指针变量指向结构体类型变量或数组(或数组元素)的起始地址。

例如

```
struct student
{
    int num;
    char name[10];
    char gender;
    int age;
    float score;
};
struct student a, st2, st[10], *p;
```

令 $p = \&a$,

st1.num
(*p).num
p->num
p[0].num

令 $p = st$,

st[0].num
(*p).num
p->num
p[0].num

● 当结构体类型的指针变量p指向一个同类型的结构体类型变量a 后，`p=&a`，下列四种表示是等价的：

`a.成员`

`(*p).成员`

`p->成员`

`p[0].成员`

->被称为 指向运算符

执行优先级跟 `.` 成员运算符一样高，是C语言中优先级最高的运算符。

● 必须注意，当p定义为指向结构体类型数据的指针后，它不能指向结构体的某一成员。

例如，`p=&a.num;` 是错误的，因为这企图让结构体指针变量指向结构体变量a中类型为int的成员num。

```
int *q;
```

```
q=&a.num;
```

是正确的，因为q和num是同一种类型 `int`。

11.3.2 结构体类型指针作为函数参数


- 结构体类型指针可以指向结构体类型的变量,因此,当形参是结构体类型指针变量时,实参也可以是结构体类型指针(即地址)。
- 结构体类型**形参指针**与结构体类型**实参指针**指向的是同一个存储空间。

【例11-7】 设有下列C程序:

```
#include<stdio.h>
struct student
{
    int num;
    char name[10];
    char gender;
    int age;
    float score;
};
```

```
void change(struct student *t)
{ printf("t=%6d%8s%3c%4d%7.2f\n",
    t->num, t->name, t->gender, t->age,
    t->score);
  t->score=95.0;
  printf("t=%6d%8s%3c%4d%7.2f\n",
    t->num, t->name, t->gender, t->age,
    t->score);
}
```

```
main( )  
{ struct student st={101,"Zhang",'M',19,89.0};  
  printf("st=%6d%8s%3c%4d%7.2f\n",  
        st.num, st.name, st.gender, st.age, st.score);  
  change(&st);  
  printf("st=%6d%8s%3c%4d%7.2f\n",  
        st.num, st.name, st.gender, st.age, st.score);  
}
```



程序运行结果为

st=	101	Zhang	M	19	89.00
t=	101	Zhang	M	19	89.00
t=	101	Zhang	M	19	95.00
st=	101	Zhang	M	19	95.00

若取结构体的地址做实参，函数改变了结构体成员的值。

因为实参结构体st是将其首地址传到形参t上，*t 与 st 等价！

- 结构体类型指针也可以指向数组或数组元素，因此，当形参是结构体类型指针变量时，实参也可以是结构体类型数组名或数组元素的地址。因此，在用结构体类型数组名作函数参数时，实际上也可以用指向结构体类型数组或数组元素的指针作为函数的参数。

与标准基本数据类型的数组与指针一样，在结构体类型数组指针作函数参数时，也可以有以下四种情况：

- 实参与形参都用结构体类型数组名。
- 实参用结构体类型数组名, 形参用结构体类型指针变量。
- 实参与形参都用结构体类型指针变量。
- 实参用结构体类型指针变量, 形参用结构体类型数组名。

【例11-8】

```
#define STU struct student
STU
{ int num;
  char name[10];
  char gender;
  int age;
  float score[3];
};
void p(STU *t, int n)
{ int k;
  for (k=0; k<=n-1; k++)
    t[k].score[2]=(t[k].score[0] + t[k].score[1])/2;
}
```

p函数还可以写作:

```
void p(STU *t, int n)
{ int k;
  for (k=0; k<=n-1; k++, t++)
    t->score[2]=(t->score[0] + t->score[1])/2;
}
```

t++使得每次循环, t向前移动一个结构体的位置。

或写作:

```
void p(STU t[ ], int n)
{ int k;
  for (k=0; k<=n-1; k++)
    t[k].score[2]=(t[k].score[0]+t[k].score[1])/2;
}
```

```
#include <stdio.h>
main( )
{ int k;
  STU stu[10]=
  {{ 101, "Zhang", 'M', 19, 95.0, 64.0 },
   { 102, "Wang", 'F', 18, 92.0, 97.0 },
   { 103, "Zhao", 'M', 19, 85.0, 78.0 },
   { 104, "Li", 'M', 20, 96.0, 88.0 },
   { 105, "Gou", 'M', 19, 91.0, 96.0 },
   { 106, "Lin", 'M', 18, 93.0, 78.0 },
   { 107, "Ma", 'F', 18, 98.0, 97.0 },
   { 108, "Zhen", 'M', 21, 89.0, 93.0 },
   { 109, "Xu", 'M', 19, 88.0, 90.0 },
   { 110, "Mao", 'F', 18, 94.0, 90.0 }
};
```

```
for (k=0; k<=9; k++)
    printf("%-8d%-10s%-5c%-6d%-7.2f%-7.2f%-7.2f\n",
           stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
           stu[k].score[0], stu[k].score[1], stu[k].score[2]);
printf("-----\n");
p(stu,10);
for (k=0; k<=9; k++)
    printf("%-8d%-10s%-5c%-6d%-7.2f%-7.2f%-7.2f\n",
           stu[k].num, stu[k].name, stu[k].gender, stu[k].age,
           stu[k].score[0], stu[k].score[1], stu[k].score[2]);
}
```


程序运行结果:

```
C:\WINDOWS\system32\cmd. × + v
```

101	Zhang	M	19	95.00	64.00	0.00
102	Wang	F	18	92.00	97.00	0.00
103	Zhao	M	19	85.00	78.00	0.00
104	Li	M	20	96.00	88.00	0.00
105	Gou	M	19	91.00	96.00	0.00
106	Lin	M	18	93.00	78.00	0.00
107	Ma	F	18	98.00	97.00	0.00
108	Zhen	M	21	89.00	93.00	0.00
109	Xu	M	19	88.00	90.00	0.00
110	Mao	F	18	94.00	90.00	0.00

101	Zhang	M	19	95.00	64.00	79.50
102	Wang	F	18	92.00	97.00	94.50
103	Zhao	M	19	85.00	78.00	81.50
104	Li	M	20	96.00	88.00	92.00
105	Gou	M	19	91.00	96.00	93.50
106	Lin	M	18	93.00	78.00	85.50
107	Ma	F	18	98.00	97.00	97.50
108	Zhen	M	21	89.00	93.00	91.00
109	Xu	M	19	88.00	90.00	89.00
110	Mao	F	18	94.00	90.00	92.00

请按任意键继续. . . |

【例11-9】结构体选择排序

```
#define STU struct student
STU
{   int num;
    char name[8];
    char gender;
    int age;
    double score;
};
```

注意: `p[j]->score`
等价于: `(*p[j]).score`

```
void sort(STU *p[], int n)
{   int i, j, k;
    STU *w;
    for (i=0; i<n-1; i++)
    {   k=i;
        for (j=i+1; j<n; j++)
            if (p[j]->score < p[k]->score)
                k=j;
        if (k != i)
            { w=p[i]; p[i]=p[k]; p[k]=w; }
    }
    return;
} /* 按成绩由低到高排序 */
```

注意: 这里交换的是指向结构体的指针,不是结构体的内容。

```
#include <stdio.h>
```

```
main( )
```

```
{ int i;
```

```
    STU stu[10]={ {101,"Zhang",'M',19,95.6},  
                  {102,"Wang",'F',18,92.4},  
                  {103,"Zhao",'M',19,85.7},  
                  {104,"Li",'M',20,96.3},  
                  {105,"Gou",'M',19,90.2},  
                  {106,"Lin",'M',18,91.5},  
                  {107,"Ma",'F',17,98.7},  
                  {108,"Zhen",'M',21,90.1},  
                  {109,"Xu",'M',19,89.8},  
                  {110,"Mao",'F',18,94.9}  
    };
```



```
STU *p[10]; /* 指针数组的每一个指针指向每一个结构体元素 */
for (i=0; i<=9; i++)
    p[i]=&stu[i];
printf("\n");
printf("No.   Name   Gender Age   Score\n");
for (i=0; i<=9; i++)
    printf("%-8d%-9s%-8c%-8d%-5.2f\n",
        p[i]->num, p[i]->name, p[i]->gender, p[i]->age, p[i]->score);
printf("-----\n");
sort(p,10);
printf("No.   Name   Gender Age   Score\n");
for (i=0; i<=9; i++)
    printf("%-8d%-9s%-8c%-8d%-5.2f\n",
        p[i]->num, p[i]->name, p[i]->gender, p[i]->age, p[i]->score);
printf("\n");
}
```

运行结果

```
C:\WINDOWS\system32\cmd. × + ▾  
  
No.    Name    Gender  Age    Score  
101    Zhang   M       19     95.60  
102    Wang    F       18     92.40  
103    Zhao    M       19     85.70  
104    Li      M       20     96.30  
105    Gou     M       19     90.20  
106    Lin     M       18     91.50  
107    Ma      F       17     98.70  
108    Zhen    M       21     90.10  
109    Xu      M       19     89.80  
110    Mao     F       18     94.90  
-----  
No.    Name    Gender  Age    Score  
103    Zhao    M       19     85.70  
109    Xu      M       19     89.80  
108    Zhen    M       21     90.10  
105    Gou     M       19     90.20  
106    Lin     M       18     91.50  
102    Wang    F       18     92.40  
110    Mao     F       18     94.90  
101    Zhang   M       19     95.60  
104    Li      M       20     96.30  
107    Ma      F       17     98.70  
  
请按任意键继续. . . |
```

这里交换的是指向结构体数组的指针数组, 结构体数组没有改变:

```
STU *p[10]; /* 指针数组的每一个指针指向每一个结构体元素 */
for (i=0; i<=9; i++)
    p[i]=&stu[i];
printf("\n");
printf("No.   Name   Gender Age   Score\n");
for (i=0; i<=9; i++)
    printf("%-8d%-9s%-8c%-8d%-5.2f\n",
        p[i]->num, p[i]->name, p[i]->gender, p[i]->age, p[i]->score);
printf("-----\n");
sort(p,10);
printf("No.   Name   Gender Age   Score\n");
for (i=0; i<=9; i++)
    printf("%-8d%-9s%-8c%-8d%-5.2f\n",
        p[i]->num, p[i]->name, p[i]->gender, p[i]->age, p[i]->score);
printf("-----\n"); /* 增加打印结构体数组 */
printf("No.   Name   Gender Age   Score\n");
for (i=0; i<=9; i++)
    printf("%-8d%-9s%-8c%-8d%-5.2f\n",
        stu[i].num, stu[i].name, stu[i].gender, stu[i].age, stu[i].score);
}
```

运行结果：中间打印的是指针数组,最后打印的是结构体数组：

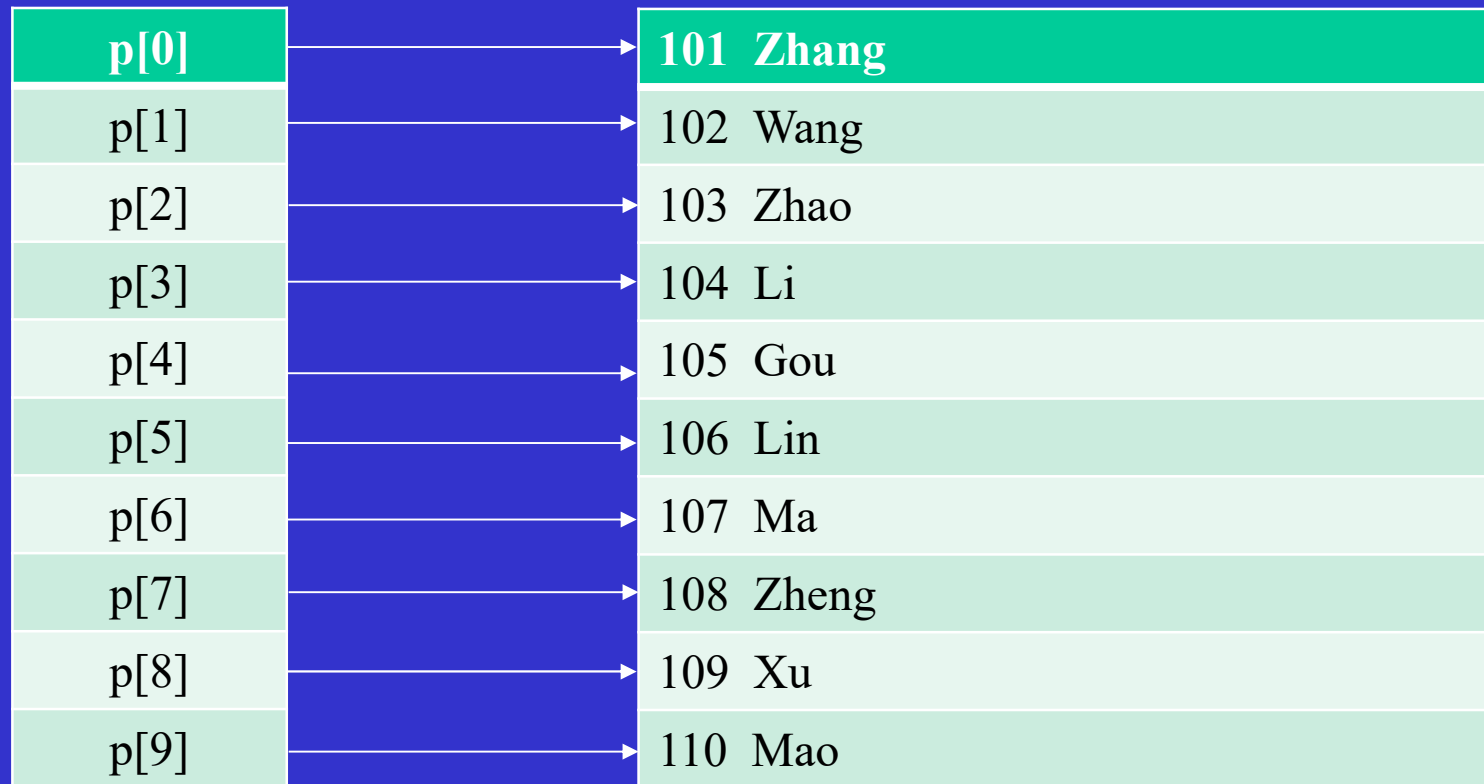
```
C:\WINDOWS\system32\cmd. X + v
```

No.	Name	Gender	Age	Score
101	Zhang	M	19	95.60
102	Wang	F	18	92.40
103	Zhao	M	19	85.70
104	Li	M	20	96.30
105	Gou	M	19	90.20
106	Lin	M	18	91.50
107	Ma	F	17	98.70
108	Zhen	M	21	90.10
109	Xu	M	19	89.80
110	Mao	F	18	94.90

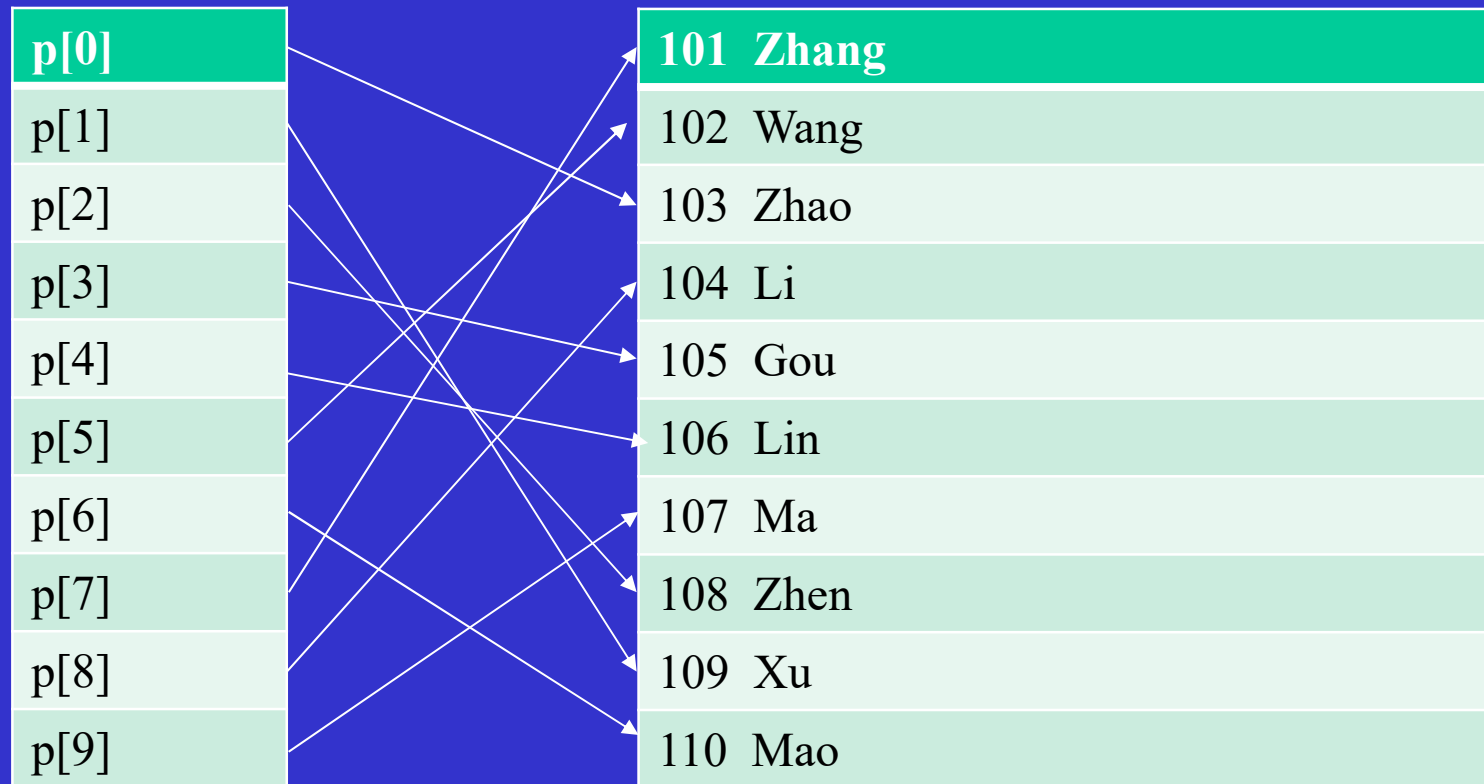
103	Zhao	M	19	85.70
109	Xu	M	19	89.80
108	Zhen	M	21	90.10
105	Gou	M	19	90.20
106	Lin	M	18	91.50
102	Wang	F	18	92.40
110	Mao	F	18	94.90
101	Zhang	M	19	95.60
104	Li	M	20	96.30
107	Ma	F	17	98.70

101	Zhang	M	19	95.60
102	Wang	F	18	92.40
103	Zhao	M	19	85.70
104	Li	M	20	96.30
105	Gou	M	19	90.20
106	Lin	M	18	91.50
107	Ma	F	17	98.70
108	Zhen	M	21	90.10
109	Xu	M	19	89.80
110	Mao	F	18	94.90

请按任意键继续. . . |



排序前：指针数组p顺序指向每一个stu结构体数组元素



排序后：排序交换的是指针数组`p`中的指针，最终指针数组`p`的各个指针指向相应`stu`结构体数组元素，但`stu`结构体数组没有改变，数组元素没有移动。

11.3.3 结构体的大小与#pragma中pack的关系

- 可以用sizeof得到结构体所占内存单元的大小。

例如有结构体：

```
struct student  /* 描述学生档案成绩 */  
{ char name[10];  
  long sno;  
  char gender;  
  float score[4];  
};
```

- sizeof(struct student) 应为31 (=10+4+1+16)

```
#include <stdio.h>
struct student { /* 描述学生档案成绩*/
    char name[10];
    long sno;
    char gender;
    float score[4];
};
main( )
{ struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
  printf("sizeof(struct student)=%d\n",sizeof(struct student));
}
```

在32位VS2008编译器上运行结果:

sizeof(struct student)=36

- 结果不是31而是36的原因是因为字节**对齐**（**alignment**）问题，这种强制对齐一来简化了处理器与内存之间传输系统的设计，二来可以提升读取数据的速度。

- 因为32位VS2008编译器中，结构体student中最长的成员的是4字节，为了不出现某个结构体变量跨4字节倍数单元地址存放，默认以结构体student中最长的成员(long, float)的4字节对齐。不足4字节倍数的name后补2个字节对齐，不足4字节的gender后补3个字节对齐，因此结构体student的大小是： $12+4+4+16=36$

- 如果在程序的开头处加上#pragma pack(4)，程序改为：

```
#include <stdio.h>
#pragma pack(4) /* 4字节对齐 */
struct student { /* 描述学生档案成绩*/
    char name[10];
    long sno;
    char gender;
    float score[4];
};
main( )
{ struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
  printf("sizeof(struct student)=%d\n",sizeof(struct student));
}
```

运行结果: sizeof(struct student)=36

结果仍是36，说明原来的程序就是默认以4字节对齐。

如果程序改为:

```
#include <stdio.h>
#pragma pack(2)    /* 2字节对齐 */
struct student {    /* 描述学生档案成绩*/
    char name[10];
    long sno;
    char gender;
    float score[4];
};
main( )
{struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
    printf("sizeof(struct student)=%d\n", sizeof(struct student));
}
```

运行结果: sizeof(struct student)=32

结构体中成员以2字节对齐，不足2字节的gender后补1个字节对齐，因此结构体student的大小是：

$10+4+2+16=32$

```
#include <stdio.h>
#pragma pack(1)    /* 1字节对齐 */
struct student {    /* 描述学生档案成绩*/
    char name[10];
    long sno;
    char gender;
    float score[4];
};
main( )
{struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
  printf("sizeof(struct student)=%d\n", sizeof(struct student));
}
```

运行结果: sizeof(struct student)=31 (10+4+1+16)

结构体中成员按照1字节对齐, 才得到了期望的31。

```
#include <stdio.h>
#pragma pack(3)    /* 会是什么结果? ? ? ? ? ? ? ? */
struct student { /* 描述学生档案成绩*/
    char name[10];
    long sno;
    char gender;
    float score[4];
};
main( )
{struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
  printf("sizeof(struct student)=%d\n",sizeof(struct student));
}
```

编译结果:

warning C4086: 杂注参数应为"1"、"2"、"4"、"8"或者 "16"

说明#pragma pack的对齐方式应该是: 1,2,4,8,16

也就是 $2^k(k \geq 0)$ 字节

运行结果: sizeof(struct student)=36
编译器按缺省方式4字节对齐。

思考题1:

```
#include <stdio.h>
struct student { /* 描述学生档案成绩*/
    char name[10];
    long sno;
    char gender;
    double score[4]; /* float 改为 double */
};
main( )
{ struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
  printf("sizeof(struct student)=%d\n",sizeof(struct student));
}
运行结果是什么?    sizeof(struct student)=56
```

切记：结构体总是默认以结构体中最长的成员类型的字节数对齐。

修改程序，打印每一个成员在内存中的起始地址：

```
#include <stdio.h>
```

```
struct student { /* 描述学生档案成绩*/
```

```
    char name[10];
```

```
    long sno;
```

```
    char gender;
```

```
    double score[4];
```

```
};
```

```
main( )
```

```
{ struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
```

```
    printf("sizeof(struct student)=%d\n",sizeof(struct student));
```

```
    printf(" name=%p\n", stu.name);
```

```
    printf("  sno=%p\n", &stu.sno);
```

```
    printf("gender=%p\n", &stu.gender);
```

```
    printf("score=%p\n", stu.score);
```

```
}
```

sizeof(struct student)=56

name=0012FF28

sno=0012FF34

gender=0012FF38

score=0012FF40

请按任意键继续...

(说明: 不同系统不同时刻的运行结果都会不同)

name[0]	
name[8]	sno
gender	
score[0]	
score[1]	
score[2]	
score[3]	

name[9]后空2个字节

sno在第2个8字节内存单元的后4个字节

gender后空7个字节

注意: 结构体成员存放时, 总是以成员数据单元大小的整倍数作为起始位置。但不同编译器结果可能不同。

思考题2:

```
#include <stdio.h>
struct student { /* 描述学生档案成绩*/
    char name[10];
    char gender;    /* sno和gender交换位置 */
    long sno;
    double score[4];
};
main( )
{ struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
  printf("sizeof(struct student)=%d\n",sizeof(struct student));
}
运行结果是什么?  sizeof(struct student)=48
```

修改程序，打印每一个成员的起始地址：

```
#include <stdio.h>
struct student { /* 描述学生档案成绩*/
    char name[10];
    char gender;
    long sno;
    double score[4];
};
main( )
{ struct student stu={"Zhang",101,'M', 98, 95, 86, 90};
  printf("sizeof(struct student)=%d\n",sizeof(struct student));
  printf(" name=%p\n", stu.name);
  printf("gender=%p\n", &stu.gender);
  printf("  sno=%p\n", &stu.sno);
  printf("score=%p\n", stu.score);
}
```

sizeof(struct student)=48

name=0012FF28

gender=0012FF32

sno=0012FF34

score=0012FF38

请按任意键继续...

(说明: 不同系统不同时刻的运行结果都会不同)

name[0]	
name[8]	gender sno
score[0]	
score[1]	
score[2]	
score[3]	

gender紧跟在name[9]后

gender后空1个字节

sno在第2个8字节内存单元的后4个字节

再次验证了: 结构体成员存放时, 总是以成员数据单元大小的整倍数作为起始位置。

第10次作业(第11章)

p.315-317 习题 1, 2, 11, 12* (选做)

说明：本章作业下次课后做也可以