



第10章 结构体和共用体

——向函数传递结构体



哈尔滨工业大学

苏小红

sxh@hit.edu.cn

向函数传递结构体

- 向函数传递结构体的**单个成员**
 - * 复制单个成员的内容
- 向函数传递结构体的**完整结构**
 - * 复制结构体的所有成员
- 向函数传递结构体的**首地址**
 - * 仅复制一个地址值

结构体变量作函数参数

```
int main()
{
    POINT position = {0, 0, 0};
    printf("Before:%d,%d,%d\n",
           position.x, position.y, position.z);
    Func(position);
    printf("After:%d,%d,%d\n",
           position.x, position.y, position.z);
    return 0;
}
```

Before:0,0,0

After:0,0,0

复制结构体的所有成员给函数

函数对结构体内容的修改不影响原结构体

`p = position;`

`p.x = position.x;`

`p.y = position.y;`

`p.z = position.z;`

```
typedef struct point
{
    int x;
    int y;
    int z;
} POINT;
void Func(POINT p)
{
    p.x = 1;
    p.y = 1;
    p.z = 1;
}
```

结构体指针作函数参数

```
int main()  
{  
    POINT position = {0, 0, 0};  
    printf("Before:%d,%d,%d\n",  
           position.x, position.y, position.z);  
    Func(&position);  
    printf("After:%d,%d,%d\n",  
           position.x, position.y, position.z);  
    return 0;  
}
```

```
pt = &position;
```

```
position.x = 1;  
position.y = 1;  
position.z = 1;
```

```
typedef struct point  
{  
    int x;  
    int y;  
    int z;  
}POINT;  
void Func(POINT *pt)  
{  
    pt->x = 1;  
    pt->y = 1;  
    pt->z = 1;  
}
```

向函数传递结构体变量的地址

Before:0,0,0

After:1,1,1

函数对结构体的修改影响原结构体

结构体变量做函数返回值

返回结构体变量也可得到修改的结构体内容，但效率低

```
int main()
{
    POINT position = {0, 0, 0};
    printf("Before:%d,%d,%d\n",
           position.x, position.y, position.z);
    position = Func(position);
    printf("After:%d,%d,%d\n",
           position.x, position.y, position.z);
    return 0;
}
```

Before:0,0,0

After:1,1,1


```
typedef struct point
{
    int x;
    int y;
    int z;
} POINT;
POINT Func(POINT p)
{
    p.x = 1;
    p.y = 1;
    p.z = 1;
    return p;
}
```

用 `const` 保护结构体指针指向的结构体

```
int main()
{
    POINT position = {0, 0, 0};
    printf("Before:%d%d%d\n",
           position.x, position.y, position.z);
    Func(&position);
    printf("After:%d%d%d\n",
           position.x, position.y, position.z);
    return 0;
}
```

```
error: assignment of member 'x' in read-only object
error: assignment of member 'y' in read-only object
error: assignment of member 'z' in read-only object
```

```
typedef struct point
{
    int x;
    int y;
    int z;
}POINT;
void Func(const POINT *pt)
{
    pt->x = 1;
    pt->y = 1;
    pt->z = 1;
}
```



结构体的一个重要应用——封装函数参数

结构体数组作函数
参数计算n个学生m
门课程的平均分

```
void AverScore(STUDENT stu[], float aver[], int n, int m)
{
    int i, j, sum[N];
    for (i=0; i<n; i++)
    {
        sum[i] = 0;
        for (j=0; j<m; j++)
        {
            sum[i] = sum[i] + stu[i].score[j];
        }
        aver[i] = (float)sum[i]/m;
    }
}
```

```
typedef struct student
{
    long    studentID;
    char    studentName[10];
    char    studentSex;
    DATE    birthday;
    int     score[4];
    float   aver;
}STUDENT;
```

```
void AverScore(STUDENT stu[], int n, int m)
{
    int i, j, sum[N];
    for (i=0; i<n; i++)
    {
        sum[i] = 0;
        for (j=0; j<m; j++)
        {
            sum[i] = sum[i] + stu[i].score[j];
        }
        stu[i].aver = (float)sum[i]/m;
    }
}
```

结构体的一个重要应用——封装函数参数

```
void AverScore(STUDENT stu[], int sum[], int n, int m)
{
    int i, j;
    for (i=0; i<n; i++)
    {
        sum[i] = 0;
        for (j=0; j<m; j++)
        {
            sum[i] = sum[i] + stu[i].score[j];
        }
        stu[i].aver = (float)sum[i]/m;
    }
}
```

```
typedef struct student
{
    long    studentID;
    char    studentName[10];
    char    studentSex;
    DATE    birthday;
    int     score[4];
    float   aver;
}STUDENT;
```

结构体数组作函数参数计
算n个学生m门课程的**总分**
和**平均分**

```
void AverScore(STUDENT stu[], int n, int m)
{
    int i, j, sum[N];
    for (i=0; i<n; i++)
    {
        sum[i] = 0;
        for (j=0; j<m; j++)
        {
            sum[i] = sum[i] + stu[i].score[j];
        }
        stu[i].aver = (float)sum[i]/m;
    }
}
```


结构体的一个重要应用——封装函数参数

```
void AverScore(STUDENT stu[], int sum[], int n, int m)
{
    int i, j;
    for (i=0; i<n; i++)
    {
        sum[i] = 0;
        for (j=0; j<m; j++)
        {
            sum[i] = sum[i] + stu[i].score[j];
        }
        stu[i].aver = (float)sum[i]/m;
    }
}
```

```
typedef struct student
{
    long    studentID;
    char    studentName[10];
    char    studentSex;
    DATE    birthday;
    int     score[4];
    float   aver;
    int     sum;
}STUDENT;
```

用**结构体类型**封装函数参数
的好处是什么？

精简参数个数
使函数接口更简洁
可扩展性好



```
void AverScore(STUDENT stu[], int n, int m)
{
    int i, j;
    for (i=0; i<n; i++)
    {
        stu[i].sum = 0;
        for (j=0; j<m; j++)
        {
            stu[i].sum = stu[i].sum + stu[i].score[j];
        }
        stu[i].aver = (float)stu[i].sum/m;
    }
}
```

小结

- 如何向函数传递结构体这样的大数据对象

向函数传递结构体的完整结构	向函数传递结构体的首地址
用结构体变量作函数参数	用结构体数组/结构体指针作函数参数
复制整个结构体成员的内容，一组数据	仅复制结构体的首地址，一个数据
参数传递直观，但开销大，效率低	参数传递效率高
函数内对结构内容的修改不影响原结构体	可修改结构体指针所指向的结构体的内容

讨论题

- 如果将下面函数的第一个形参 **STUDENT stu[]** 修改为指向结构体数组的指针，即 **STUDENT *pt**，那么程序该如何修改？

```
void AverScore(STUDENT stu[], int n, int m)
{
    int i, j;
    for (i=0; i<n; i++)
    {
        stu[i].sum = 0;
        for (j=0; j<m; j++)
        {
            stu[i].sum = stu[i].sum + stu[i].score[j];
        }
        stu[i].aver = (float)stu[i].sum/m;
    }
}
```



