

结构体和共用体

2020年10月23日 15:52

- **声明结构体类型**

```
struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    int yearofBirth;
    int score[4];
};
```

此时并没有定义任何结构体变量，编译器不为其分配内存。

- **结构变量的定义**

方法1：

先定义结构体类型，再定义变量名：

```
struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    int yearofBirth;
    int score[4];
};
struct student stu1;
```

方法2：

定义结构体类型的同时定义结构体变量

```
struct student
{
    long studentID;
    char studentName[10];
    char studentSex;
    int yearofBirth;
    int score[4];
}stu1; //放在分号之前
```

方法3：—

直接定义结构体变量。不指定结构体标签

```
struct
```

```

{
—long studentID;
—char studentName[10];
—char studentSex;
—int yearofBirth;
—int score[4];
}stu1; (不好用)

```

- 为结构体命名的方法:

- 1.使用结构体标签
- 2.用**typedef**来给数据类型定义一个别名。

```
struct student
```

```

{
    long studentID;
    char studentName[10];
    char studentSex;
    int yearofBirth;
    int score[4];
};

```

```
typedef struct student STUDENT;
```

typedef作用:

使用typedef为现有类型创建别名，给变量定义一个易于记忆且意义明确的新名字。例如:

```
typedef unsigned int UINT
```

也可以:

```
typedef struct student //此时结构体标签可有可无
```

```

{
    long studentID;
    char studentName[10];
    char studentSex;
    int yearofBirth;
    int score[4];
}STUDENT;

```

此后，STUDENT 与struct student为同义词

- 结构体变量的初始化

在定义的同时初始化:

```
STUDENT stu1={10030897, "王刚", 'M' , 1991, {72, 83, 90, 82}};
```

或

```
struct student stu1={10030897, "王刚", 'M' , 1991,
{72, 83, 90, 82}};
```

数据类型需要一致，必须是常量，不能是变量，用逗号分隔。

- **结构体与数组的嵌套**

```
struct student
```

```
{
```

```
    long studentID;
```

```
    char studentName[10];
```

```
    char studentSex;
```

```
    int yearofBirth;
```

```
    int score[4];
```

```
}stu[30];
```

一个结构体也可以嵌套在另一个结构体里面

嵌套结构体：在一个结构体内包含了另一个结构体作为成员

- **结构体数组的定义和初始化**

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

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

```
STUDENT stu[30] = {{100310121, "王刚", 'M', {1991,5,19},{72,83,90,82}},
                   {100310122, "李小明", 'M', {1992,8,20},{88,92,78,78}},
                   {100310123, "王丽红", 'F', {1991,9,19},{98,72,89,66}},
                   {100310124, "陈莉莉", 'F', {1992,3,22},{87,95,78,90}}
                   };
```

- **结构体所占内存的字节数**

不是所有成员占内存的总和。

用sizeof获得。

```
printf ( "%d\n",sizeof (stu1) ) ; 或者
```

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

```
printf ( "%d\n",sizeof (STUDENT) ) ;
```

要大于所有成员内存和，原因：

内存对齐(Memory-alignment)

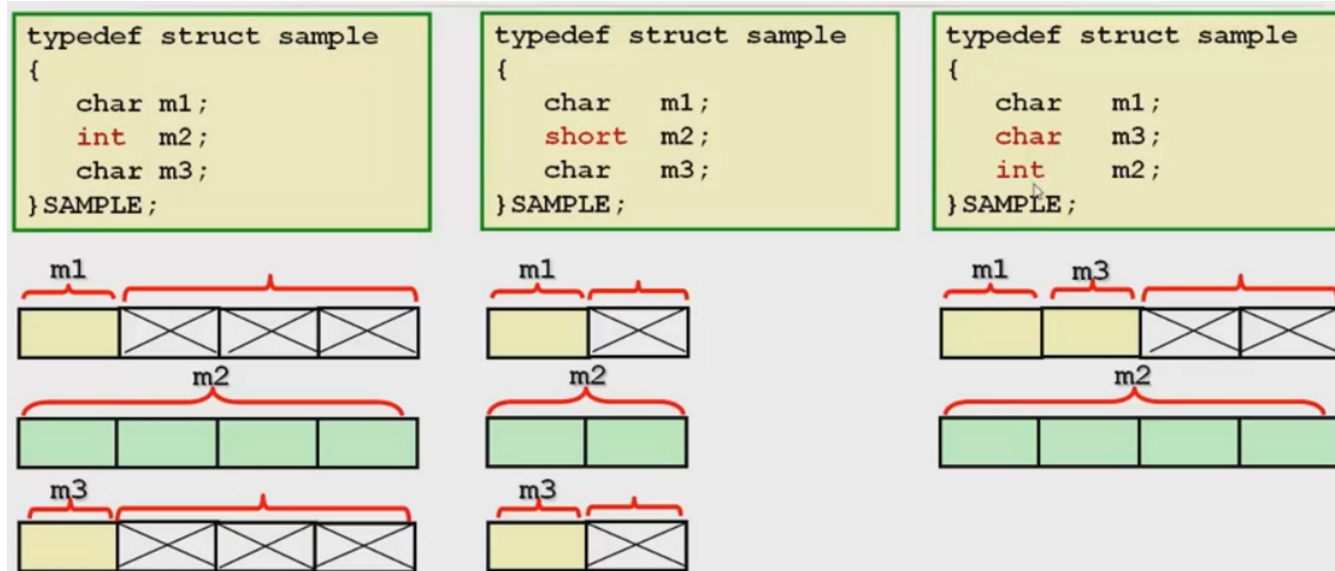
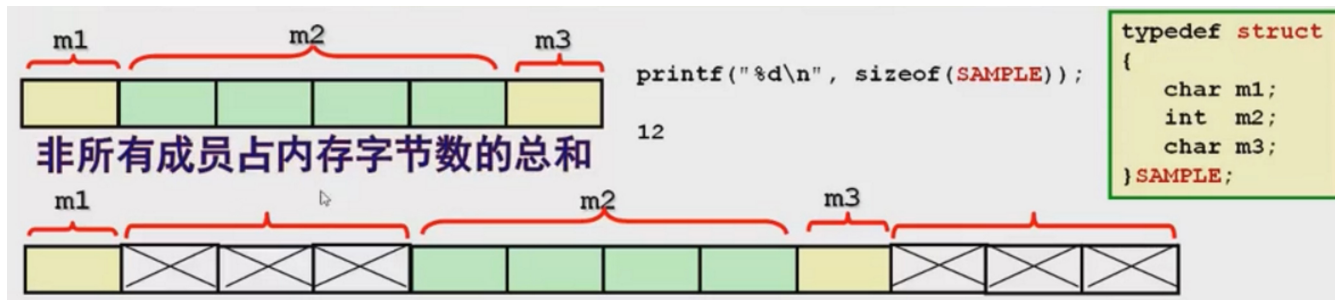
对于大多数计算机，数据项要求从某个数量字节的倍数开始存放

short型数据从偶数地址开始存放，而int型数据则被对齐在4字节地址边界

为了满足内存地址对齐的要求，需要在较小的成员后加入补位

用sizeof关键字求结构体长度时，返回的是最大基本类型所占字节的整数倍

如以下结构体的大小为12字节：



-什么要求内存地址对齐呢？

-提高内存寻址效率

32位体系结构中,int值被对齐在4字节地址边界

读写一个4字节int型数据，只需一次内存访问操作

结构体在内存中所占的字节数不仅与所定义的结构体类型有关，还与计算机系统本身有关

不同的系统和编译器，内存对齐方式可能会不同，是机器相关的。

一定要用sizeof（）运算符

• 访问结构体成员

成员选择运算符（圆点运算符） . 一个圆点。

结构体变量.成员名

如：stu1.studentName

stu1.studentID = 12213432

而字符数组不支持直接用字符串赋值：~~stu1.studentName = "周毅"~~

//stu1.studentName代表首地址

可以用：strcpy (stu1.studentName,"周毅")；

- **对嵌套的结构体成员,必须以级联方式访问**

```
stu1.birthday . year = 1991 ;  
stu1.birthday.month = 5;  
stu1.birthday . day = 19;
```

- **结构体赋值操作**


可以使用`stu2 = stu1` ; 来使两个结构体变量相同

不能使用==和!=来判定两个结构体相等或不等

`if(stu1==stu2)`

把数组放到一个“空”的结构体内封装以后, 就可以直接复制数组了

```
typedef struct  
{  
    int member[5];  
}ARRAY;  
ARRAY a = {1,2,3,4,5};  
ARRAY b;  
b = a;
```



- **结构体变量的取地址操作**

```
int main()  
{  
    STUDENT stu1, stu2;  
    int i;  
    printf("Input a record:\n");  
    scanf("%ld", &stu1.studentID);  
    scanf("%s", stu1.studentName); /* 输入学生姓名, 无需加& */  
    scanf(" %c", &stu1.studentSex); /* %c 前有一个空格 */  
    scanf("%d", &stu1.birthday.year);  
    scanf("%d", &stu1.birthday.month);  
    scanf("%d", &stu1.birthday.day);  
    for (i=0; i<4; i++)  
    {  
        scanf("%d", &stu1.score[i]);  
    }  
}
```

学生姓名存在于**字符数组**, 因此不需要&

加空格是因为**输入学生姓名之后键入了一个回车符**, 防止下一个scanf将其读入。

- **结构体指针定义**

`STUDENT stu1;`

`STUDENT *Pt;` //定义一个指向结构体变量的指针pt

`pt = &stu1` //指针初始化, 指向结构体变量

也可以定义的同时初始化

```
STUDENT *pt = &stu1
```

- **访问结构体指针指向的结构体成员**

1. `(*pt).studentID = 1`

指针解引用 成员选择运算符

2. **指向运算符**

```
pt -> studentID = 1
```

- **当结构体嵌套时**

级联方式: `stu1.birthday.year=2002`

同理:

```
(*pt).birthday.year=2002
```

```
pt -> birthday.year=2002
```

- **结构体数组的指针**

```
STUDENT stu[30];
```

```
STUDENT *pt;
```

```
pt=stu
```

等价于:

```
STUDENT *pt = stu
```

等价于:

```
STUDENT *pt = &stu[0];
```

都是让数组指向了stu数组的首地址

- **结构体数组的指针访问数组成员**

此时已经指向了 `&stu[0]`

访问:

```
pt -> studentID;
```

等价于

```
(*pt).studentID;
```

等价于

```
stu[0].studentID;
```

此时`stu[i]`相当于结构体变量

- **`pt++`是让pt指向下一个数组元素**

- **共用体**

和结构体一样, 都是用户自定义的数据类型

结构体(struct): 把关系紧密且逻辑相关的多种不同类型的的变量,组织到一个统

一的名字之下

共用体，也称**联合** (union): 把**情形互斥但逻辑相关**的多种不同类型的变量，组织到一个统一的名字之下

- **共用体声明**

```
struct sample          union sample
{
    short i;            {
    char ch ;           short i;
    float f ;           char ch ;
                        float f ;
};                      };
```

极其相似

```
struct sample s = {1, 'A', 3.14};
```

而**union sample u = {1};** **//花括号不能省略**

只能为第一个成员进行初始化

- **共用体内存分配**

所有共用体成员**彼此覆盖，共享同一存储空间**。

所占字节数取决于占空间最多的那个成员变量。

- **共用体访问**

同样使用成员选择运算符

```
union sample u;
```

```
u.i = 1;
```

```
u.ch = 'A';
```

```
u.f = 3.14;
```

每次只能保存一个成员的值，最终起作用的是最后一次赋值的成员。

- **共用体的应用**

1. 节省储存空间

为共用体添加标记字段

姓名	性别	年龄	婚姻状况						婚姻状况 标记
			未婚	已婚			离婚		
				结婚日期	配偶姓名	子女数量	离婚日期	子女数量	

```
struct person
{
    char name[20];
    char sex;
    int age;
    union maritalState marital;
    int marryFlag; //婚姻状态标记字段
};
struct person p1;
```

每次对共用体的成员赋值时，程序负责改变标记字段的内容

C语言程序设计

共用体的一个主要问题：如何标记共用体中当前起作用的成员是哪一个？

```
if (p1.marryFlag == 1)
{
    //未婚
}
else if (p1.marryFlag == 2)
{
    //已婚
}
else
{
    //离婚
}
```

2.构造混合的数据类型

构造储存混合数据类型的数组

typedef union

```
{
    int i;
    float f;
}NUMBER;
```

NUMBER array [100];

array[0].i=10;

array[1].f=3.14;

此时每个NUMBER类型的数组array的数组元素都有两个成员，既可储存int型数据，也可以储存float型数据。