西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

复杂数据在C程序中的表示

1、结构体类型

2、结构体对象

- C语言还支持用户自定义类型UDT（user defined type），所谓自定义类型是根据应用程序具体需要而设计的数据类型。

- 数组是一种数据形式，其特点是多个相同类型元素集合起来；结构体是另一种重要的数据形式，其特点是不同类型成员组合起来。数组和结构体形成了两种风格迥异的聚合（aggregate）方式，通过它们及其相互组合、相互嵌套的机制可以构造出复杂的、满足应用要求的自定义数据类型。

- 共用体又称联合，是一种可以共享存储空间的自定义类型，位域是以二进制位为数据形式的自定义类型，枚举类型是以整数常量聚合的自定义类型。
- 通过typedef，任何内置数据类型或自定义类型可以重新命名，进而简化了类型名称，方便形成可移植的、规范的应用程序数据类型体系。

8.1 结构体类型

- 有时需要将不同类型但又相互联系的数据组合在一起使用。这些数据项的类型是不同的，因此不能使用数组表示它们。如果分别定义为相互独立的变量，又难以反映出它们之间的内在联系，编程时数据管理工作量大且复杂。

8.1 结构体类型

- C语言的结构体允许将不同类型的数据元素组合在一起形成一种新的数据类型，其声明形式为：

```
struct 结构体类型名 {
   成员列表
};
```

8.1 结构体类型

- 结构体类型名与struct一起作为类型名称，成员列表则是该类型的数据元素的集合，数目可以任意多，由具体应用确定。一对大括号｛｝是成员列表边界符，后面必须用分号（；）结束。

8.1 结构体类型

- 结构体类型声明时必须给出各个数据成员的类型声明，其一般形式为：

```
成员类型  成员名列表;
```

- 声明时成员名列表允许为多个，用逗号（，）作为间隔。

8.1 结构体类型

- 例如可以通过如下声明建立能表示学生信息的数据类型。

```
struct tagSTUDENT {  //学生信息类型
    int no;          //声明一个整型数据成员表示学号
    char name[21];   //声明一个字符数组（字符串）数据成员表示姓名
    char sex;        //声明一个字符数据成员表示性别
    int age;         //声明一个整型数据成员年龄
    char qq[11];     //声明一个字符数组（字符串）数据成员表示QQ号
    double score;    //声明一个浮点型数据成员表示成绩
};
```

8.1 结构体类型

- 结构体类型声明一般放在程序文件开头，或者放到头文件中被程序文件包含，此时这个声明是全局的。在全局作用域内，该声明处处可见，因此同作用域内的所有函数都可以使用它。

- 结构体类型声明一般放在程序文件开头，或者放到头文件中被程序文件包含，此时这个声明是全局的。在全局作用域内，该声明处处可见，因此同作用域内的所有函数都可以使用它。

- 结构体类型声明也可以放到函数内部，此时这个声明是局部的。若在函数内部有同名的结构体类型声明，则全局声明在该函数内部是无效的，有效的是局部声明的。例如：

```
struct tagDATE { //全局声明的tagDATE
    int year,month,day;
};
void fun()
{
    struct tagDATE { //局部声明的tagDATE
        int year,month,day,week;
    }; //全局tagDATE在函数无效，有效的是局部声明的tagDATE
}
```

8.1 结构体类型

- 通常，结构体类型名都是大写，以便与C语言内置数据类型（都是小写）明显区别开来。举例中的结构体类型名除了大写外，前面均加了一个“tag”。之所以这样做是因为使用后面的typedef可以将结构体类型命名简化，那里的命名全用大写，结构体类型名应与它有所区别。例如：

```
struct tagDATE { //日期类型
    int year,month,day; //年，月，日
};
typedef struct tagDATE DATE;
//将结构体类型struct tagDATE重新命名得到简化的DATE类型名
```

8.1 结构体类型

- 结构体类型声明的补充说明。
- （1）首先，需要理解struct本身是一种抽象的数据类型。即struct笼统地代表结构体，但它究竟有哪些数据成员是不定的，因此不能直接用struct去定义变量。例如：

```
int a; //正确，int是具体的数据类型
struct b; //错误，struct是抽象的数据类型
```

- 所以，结构体使用前必须先声明结构体类型，有了这个具体的数据类型才谈得上使用这种类型。

- （2）结构体类型声明向编译器声明了一种新的数据类型，该数据类型有不同类型的数据成员。例如上述的struct tagSTUDENT，它和内置数据类型名（如int、char、double等）一样是类型名称，而不是该类型的一个实体。因此尽管成员类似变量的定义，但类型声明时并不会产生该成员的实体，即为它分配存储空间。例如：

```
struct tagCOMPLEX { //复数类型
  double r,i; //声明有两个浮点型数据成员，但不会产生实体（分配内存）
};
```

- （3）结构体类型声明时成员列表可以是任意数目、任意类型的成员，甚至是结构体类型成员，例如：

```
struct tagSTAFF { //职员信息类型
    int no; //工号，整型
    char name[21]; //姓名，字符数组（字符串）
    char sex; //性别，字符型
    struct tagDATE birthday; //出生日期，结构体类型
    double salary; //薪水，浮点型
};
```

8.1 结构体类型

- 结构体类型可以将数组和结构体这两种截然不同的数据聚合方式嵌套起来使用，从而让C语言有了表示复杂数据结构的能力。

- （4）结构体类型的一对大括号（｛｝）可以看作是一个作用域，因此其成员名称可以与外部其他标识符相同，这个特点使得结构体类型很适合数据封装。

8.2 结构体对象

- 结构体类型可以表示大型结构的数据对象，数据表示形式层次更高，因此将结构体类型的实体称为对象，区别于以前的变量。
- 定义结构体对象称为结构体类型实例化（instance），实例化会根据数据类型为结构体对象分配内存单元。

8.2.1 结构体对象的定义

- 定义结构体对象有三种形式。
- 1．先声明结构体类型再定义对象
- 假定事先已经声明了结构体类型，可以用它来定义结构体对象，即将该类型实例化。一般形式为：

```
struct 结构体类型名 结构体对象名列表;
```

- 结构体对象名列表是一个或多个对象的序列，各对象之间用逗号（，）分隔，最后必须用分号（；）结束，对象取名必须遵循标识符的命名规则

8.2.1 结构体对象的定义

- 示例

```
struct tagSTUDENT a,b; //定义结构体对象
```

8.2.1 结构体对象的定义

- 2. 声明结构体类型的同时定义对象
- 一般形式为：

```
struct 结构体类型名 {
    成员列表
} 结构体对象名列表;
```

- 这种形式需要注意结构体对象名列表是在右大括号（}）和分号（;）之间。

8.2.1 结构体对象的定义

- 示例

```
struct tagDATE { //日期类型
    int year,month,day; //年, 月, 日 整型
} d1,d2; //定义结构体对象
```

8.2.1 结构体对象的定义

- 3. 直接定义结构体对象
- 一般形式为：

```
struct {
   成员列表
} 结构体对象名列表;
```

- 这种形式显然是第二种形式的特例，即不声明结构体类型只定义结构体对象。
- 第一种形式应用得最普遍和最灵活。

8.2.1 结构体对象的定义

- 4．结构体对象的内存形式
- 实例化结构体对象后，对象会得到存储空间。如图所示为struct tagSTUDENT对象的内存结构。

图8.1 结构体对象的内存结构

no	name	sex	age	qq	score

- 结构体各成员是根据在结构体声明时出现的顺序依次分配空间的，在初始化结构体对象和使用指针操作结构体对象时尤其需要注意这个特点。

8.2.1 结构体对象的定义

- 结构体对象的内存长度是各个成员内存长度之和，推荐使用sizeof运算，由编译器自动确定内存长度，例如：

```
sizeof(struct tagSTUDENT) //得到结构体类型的内存长度
sizeof a //得到结构体对象a的内存长度
```

8.2.1 结构体对象的定义

FAQ

需要注意，在有的编译器中，sizeof得到的结构体内存长度可能比理论值大。例如VC环境下，下面两个结构体类型

```
struct A {
  int a;    //4字节
  char b;   //1字节
  short c; //2字节
};
```

```
struct B {
  char b;   //1字节
  int a;    //4字节
  short c; //2字节
};
```

成员相同（仅顺序不同），理论上它们的内存长度应是4+1+2=7。但实际上sizeof(struct A)的结果为8，sizeof(struct B)的结果为12，这是什么原因呢？

8.2.1 结构体对象的定义

FAQ

为了加快数据存取的速度，编译器默认情况下会对结构体成员和结构体本身（实际上其他数据对象也是如此）存储位置进行处理，使其存放的起始地址是一定字节数的倍数，而不是顺序存放，称为字节对齐。

8.2.1 结构体对象的定义

FAQ

设对齐字节数为 $n(n=1,2,4,8,16)$，每个成员内存长度为 L_i，$Max(L_i)$ 为最大的成员内存长度。字节对齐规则是：

①结构体对象的起始地址能够被 $Max(L_i)$ 所整除；

②结构体中每个成员相对于起始地址的偏移量，即对齐值应是 $\min(n, L_i)$ 的倍数。若不满足对齐值的要求，编译器会在成员之间填充若干个字节（internal padding）；

③结构体的总长度值应是 $\min(n, Max(L_i))$ 的倍数，若不满足总长度值的要求，编译器在为最后一个成员分配空间后，会在其后填充若干字节（trailing padding）。

8.2.1 结构体对象的定义

FAQ

例如VC默认的对齐字节数n=8，则struct A的内存长度分析如下：

（1）A的第一个成员a为int，对齐值min(n,sizeof(int))为4，成员a相对于结构体起始地址从0偏移开始，满足4字节对齐要求；第二个成员b为char，对齐值min(n,sizeof(char))为1，b紧接着a后面从偏移4开始，满足1字节对齐要求；第三个成员c为short，对齐值min(n,sizeof(short))为2，如果c紧接着b后面从偏移5开始就不满足2字节对齐要求，因此需要补充1个字节，从偏移6开始存储。结构体A的内存长度=4+1+1(补充)+2=8。

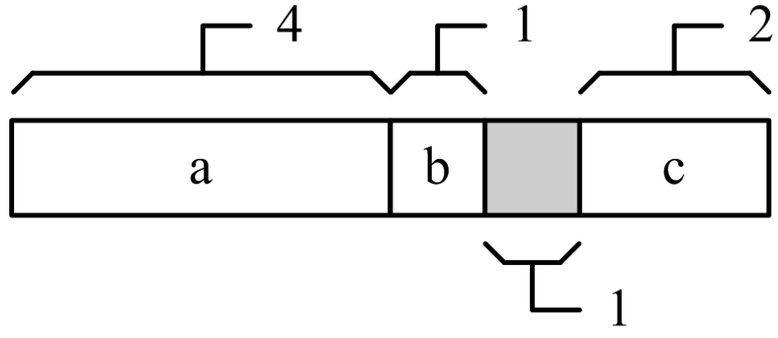
8.2.1 结构体对象的定义

FAQ

如图所示为A的内存结构，阴影部分是满足字节对齐要求而补充的字节。

```
struct A {
   int a;    //4字节
   char b;   //1字节
   short c; //2字节
};
```

图8.2 结构体字节对齐示意



（a）struct A

8.2.1 结构体对象的定义

FAQ

例如VC默认的对齐字节数n=8，则struct B的内存长度分析如下：

（2）B的第一个成员b为char，对齐值min(n,sizeof(char))为1，成员b相对于结构体起始地址从0偏移开始，满足1字节对齐要求；第二个成员a为int，对齐值min(n,sizeof(int))为4，如果a紧接着b后面从偏移1开始，不满足4字节对齐要求，因此补充3个字节，从偏移4开始存储；第三个成员c为short，对齐值min(n,sizeof(short))为2，c紧接着a后面从偏移8开始，满足2字节对齐要求。则总的内存长度=1+3(补充)+4+2=10。由于n大于最大的成员内存长度（4），故结构体长度应是4的倍数，因此最后需要再补充2个字节。结构体B的内存长度=1+3(补充)+4+2+2(补充)=12。

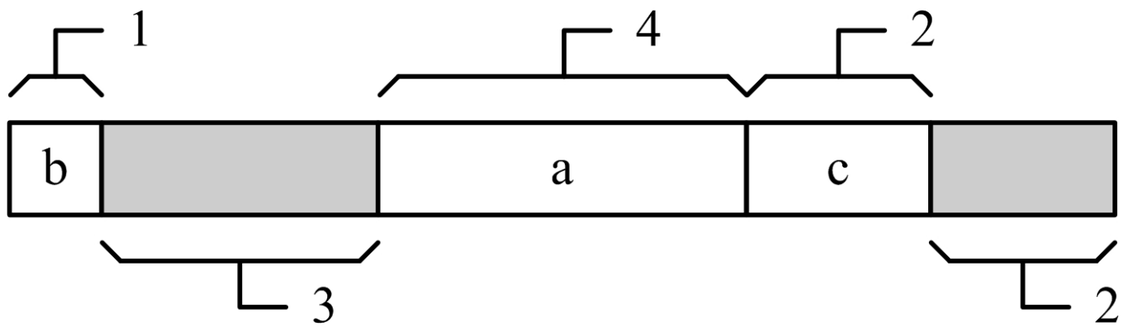
8.2.1 结构体对象的定义

FAQ

如图所示为B的内存结构，阴影部分是满足字节对齐要求而补充的字节。

```
struct B {
  char b;   //1字节
  int a;    //4字节
  short c; //2字节
};
```

图8.2 结构体字节对齐示意



（b）struct B

8.2.1 结构体对象的定义

FAQ

使用预处理命令#pragma pack(n)，可以设定对齐字节数 。例如：

```
#pragma pack(push) //保存对齐字节数
#pragma pack(1) //设定对齐字节数为1
struct A {
    int a;    //4字节
    char b;   //1字节
    short c; //2字节
};
#pragma pack(pop) //恢复对齐字节数
```

```
#pragma pack(push)
#pragma pack(1)
struct B {
    char b; //1字节
    int a;  //4字节
    short c; //2字节
};
#pragma pack(pop)
```

sizeof(struct A)和sizeof(struct B)的结果均为7。

8.2.2 结构体对象的初始化

- 可以在结构体对象定义时进行初始化。第一种定义初始化的一般形式为：

```
struct 结构体类型名 结构体对象名1={初值序列1}, ......;
```

- 第二种定义初始化的一般形式为：

```
struct 结构体类型名 {
    成员列表
} 结构体对象名1={初值序列1}, ......;
```

8.2.2 结构体对象的初始化

- 结构体对象的初值序列与数组一样，必须用一对大括号（{ }）将它括起来，即使只有一个数据也是如此。如果结构体对象嵌套了结构体成员，则该成员的初值可以用或不用大括号括起来。初值的类型和次序必须与类型声明时的一致。例如：

```
struct tagSTAFF s1={1001,"Li Min",'M',{1980,10,6},2700.0};
struct tagSTAFF s2={1002,"Ma Gang",'M',1978,3,22,3100.0};
```

8.2.3 结构体对象的使用

- 1．结构体对象成员引用
- 使用结构体对象主要是引用它的成员，其一般形式为：

```
结构体对象名.成员名
```

- 其中，小数点（．）为对象成员引用运算符。

8.2.3 结构体对象的使用

表8-1 对象成员引用运算符

运算符	功能	目	结合性	用法
.	对象成员引用运算	双目	自左向右	object.member

对象成员引用运算符在所有运算符中优先级较高，其作用是引用结构体对象中的指定成员，运算结果为左值（即成员本身），因此可以对运算结果做赋值、自增自减、取地址等运算。

8.2.3 结构体对象的使用

- 示例

```
struct tagSTAFF a,b;
a.no=10002;
//将10002赋值给a对象中的no成员，对象成员引用运算结果是左值（即成员本身）
b.salary=a.salary+500.0; //在表达式中可以引用对象成员
a.no++; //按优先级等价于(a.no)++
```

8.2.3 结构体对象的使用

- 对象成员引用运算时需要注意以下几点：
- （1）对象成员引用运算符（．）左边运算对象object必须是结构体对象，右边member必须是结构体中的成员名。

8.2.3 结构体对象的使用

- （2）如果成员本身又是一个结构体对象，就要用成员引用运算符，一级一级地引用。例如：

```
struct tagSTAFF x;
x.birthday.year=1990,x.birthday.month=5,x.birthday.day=12;
//逐级引用成员
```

8.2.3 结构体对象的使用

- 2．结构体对象输入与输出
- 不能将一个结构体对象作为整体进行输入或输出，只能对结构体对象中基本类型成员逐个进行输入或输出。例如：

```
struct tagSTAFF x;
scanf("%d%c%f",&x.no,&x.sex,&x.salary);
//整型、字符型、浮点型成员输入
scanf("%d%d%d",&x.birthday.year,&x.birthday.month,&x.birth
day.day);
gets(x.name); //字符串成员输入
```

8.2.3 结构体对象的使用

- 3．结构体对象的运算
- 结构体对象可以进行赋值运算，但不能对进行算术运算、关系运算等，例如：

```
struct tagCOMPLEX m,n,k;
m=n; //正确，结构体对象允许赋值
k=m+n; //错误，结构体对象不能做算术运算
m>n; //错误，结构体对象不能做关系运算
```

- 结构体对象赋值时，本质上是按内存形式将一个对象的全体成员完全复制到另一个对象中。如果结构体对象包含大批成员（如数组），则赋值将耗费大量运行时间。


程序设计