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

编程使用复杂数据

1、函数传递复杂数据

2、共用体类型和对象

8.5 结构体与函数

- 结构体对象、数组、指针均可以作为函数参数，函数还可以返回结构体对象和结构体指针。

8.5.1 结构体对象作为函数参数

- 将结构体对象作为函数实参传递到函数中，采用值传递方式。结构体对象内存单元的所有内容像赋值那样复制到函数形参中，形参必须是同类型的结构体对象。

8.5.1 结构体对象作为函数参数

- 示例

```
struct tagDATA {
    int data; //整型成员
    char name[10]; //数组成员
};
void fun1(struct tagDATA x); //函数原型
void fun2()
{
    struct tagDATA a={1,"LiMin"};
    fun1(a); //函数调用
}
```

8.5.1 结构体对象作为函数参数

- 函数调用时，实参对象a的data和name成员逐一复制到形参x对象中。因此这种传递方式会增加函数调用在空间、时间上的开销，特别是当结构体的长度很大时，开销会急剧增加。

8.5.1 结构体对象作为函数参数

- 采用值传递方式，形参对象仅是实参对象的一个副本，在函数中若修改了形参对象并不会影响到实参对象，即形参对象的变化不能返回到主调函数中。
- 实际编程中，传递结构体对象时需要考虑结构体的规模带来的调用开销，如果开销很大时建议不要用结构体对象作为函数参数。

8.5.1 结构体对象作为函数参数

- 将结构体数组作为函数参数，采用地址传递方式。函数调用实参是数组名，形参必须是同类型的结构体数组。

8.5.1 结构体对象作为函数参数

- 示例

```
void fun3(struct tagDATA X[]); //函数原型
void fun4()
{
    struct tagDATA A[3]={1,"LiMin",2,"MaGang",3,"ZhangKun
"};
    fun3(A); //函数调用
}
```

8.5.1 结构体对象作为函数参数

- 函数调用时，无论数组有多少个元素，每个元素（结构体对象）有多大规模，传递的参数是数组的首地址，其开销非常小。

8.5.1 结构体对象作为函数参数

- 将结构体指针作为函数参数，采用地址传递方式。函数调用实参是结构体对象的地址，形参必须是同类型的结构体指针。

8.5.2 结构体数组作为函数参数

- 示例

```
void fun5(struct tagDATA *p); //函数原型
void fun6()
{
    struct tagDATA a={1,"LiMin"};
    fun3(&a); //函数调用
}
```

8.5.2 结构体数组作为函数参数

- 函数调用时，无论结构体有多大规模，传递的参数是一个地址值，其开销非常小。

- 采用地址传递方式，在函数中若按间接引用方式修改了形参对象本质上就是修改实参对象。因此，使用结构体指针作为函数参数可以向主调函数传回变化后的结构体对象。

8.5.2 结构体数组作为函数参数

- 如果希望用结构体指针减少函数调用开销而又不允许在函数中意外修改实参对象，可以将结构体指针形参作const限定，例如：

```
void fun7(const struct tagDATA *p)
{
  p->data=100; //不能修改常对象成员
}
```

- 函数中任何试图修改形参对象的代码都会导致语法出错，进而防止意外修改。

8.5.4 函数返回结构体对象或指针

- 函数的返回类型可以是结构体类型，这时函数将返回一个结构体对象。例如：

```
struct tagDATA fun8()
{
  struct tagDATA a={1,"LiMin"};
  return a; //返回结构体对象，复制到临时对象中
}
void fun9()
{
  struct tagDATA b;
  b=fun8(); //函数返回结构体对象，并且赋值
}
```

8.5.4 函数返回结构体对象或指针

- 函数返回结构体对象时，将其内存单元的所有内容复制到一个临时对象中。因此函数返回结构体对象时也会增加调用开销。
- 函数的返回类型可以是结构体指针类型，但不要返回局部对象指针，因为它是无效的。

8.6 共用体

- 共用体（union）是一种成员共享存储空间的结构体类型。

8.6.1 共用体概念及类型声明

- 共用体类型是抽象的数据类型，因此程序中需要事先声明具体的共用体类型，一般形式为：

```
union 共用体类型名 {
   成员列表
};
```

- 共用体类型名与union一起作为类型名称，成员列表是该类型数据元素的集合。一对大括号｛｝是成员列表边界符，后面必须用分号（；）结束。

8.6.1 共用体概念及类型声明

- 共用体类型声明时必须给出各个成员的类型声明，其形式为：

```
成员类型  成员名列表;
```

- 成员名列表允许任意数目的成员，用逗号（，）作为间隔。

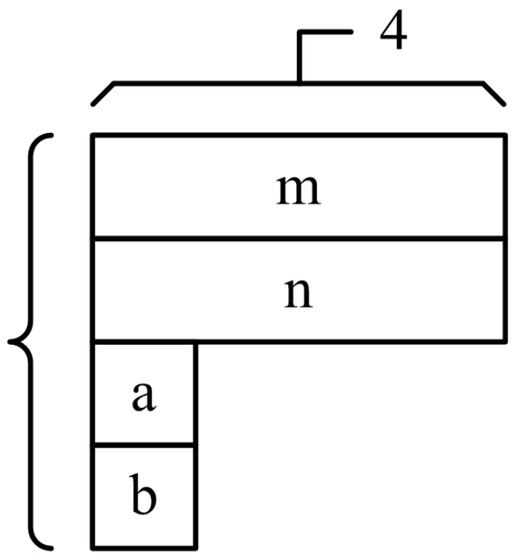
8.6.1 共用体概念及类型声明

- 共用体中每个成员与其他成员之间共享内存。如有共用体类型

```
union A {
   int m,n;   //整型成员
   char a,b; //字符成员
};
```

- m、n、a、b共享内存单元，其内存结构
- 如图所示。

图8.4 共用体内存结构示意



（a）union A

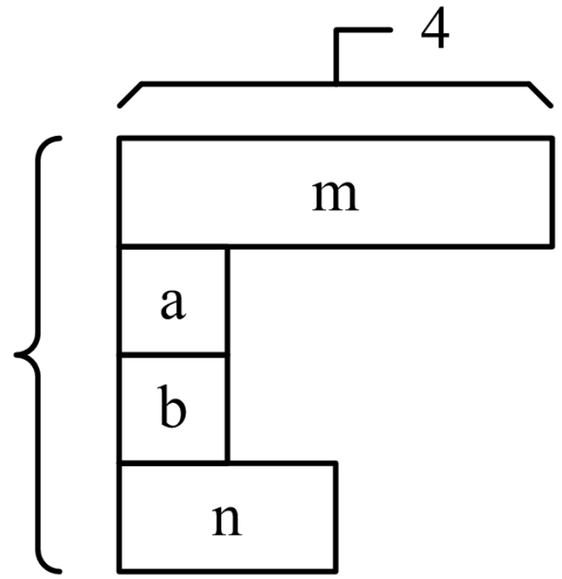
8.6.1 共用体概念及类型声明

- 如有共用体类型

```
union B {
   int m;      //整型成员
   char a,b;   //字符成员
   short n;    //短整型成员
};
```

- m、a、b、n共享内存单元，其内存结构
- 如图所示。

图8.4 共用体内存结构示意



（b）union B

8.6.1 共用体概念及类型声明

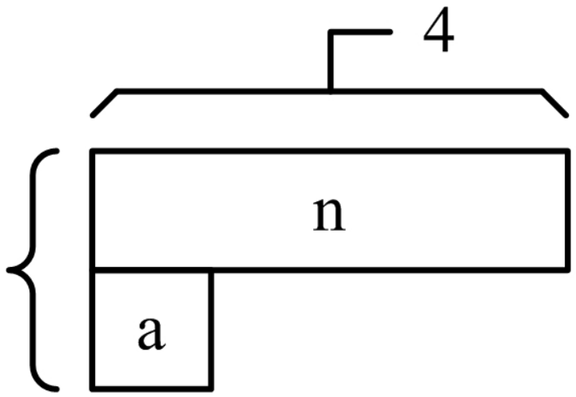
- 比较共用体和结构体类型

```
union tagUDATA { //共用体类型
  int n;  //整型成员
  char a; //字符成员
};
```

```
struct tagSDATA { //结构体类型
  int n;  //整型成员
  char a; //字符成员
};
```

8.6.1 共用体概念及类型声明

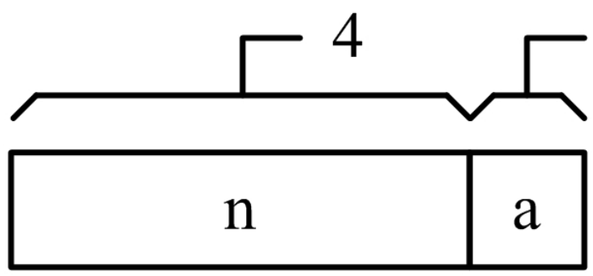
- 如图所示为union tagUDATA类型内存结构，可以看出两个成员共享了同一段内存单元，考虑到整型有4个字节，字符型只有1个字节，相当于a是n的一部分。



（a）union tagUDATA

8.6.1 共用体概念及类型声明

- 如图所示为struct tagSDATA类型内存结构，可以看出两个成员n和a是各自独立的。



（b）struct tagSDATA

8.6.1 共用体概念及类型声明

- 显然，结构体与共用体的内存形式是截然不同的。共用体内存长度是所有成员内存长度的最大值，结构体内存长度是所有成员内存长度之和。建议用sizeof取它们的内存长度。
- 需要注意，共用体内存分配时仍然采用字节对齐规则。

8.6.2 共用体对象的定义

- 与结构体对象相似，定义共用体对象也有三种形式：

①先声明共用体类型再定义共用体对象
```
union 共用体类型名 共用体对象名列表;
```
②同时声明共用体类型和定义共用体对象
```
union 共用体类型名 { 成员列表 } 共用体对象名列表;
```
③直接定义共用体对象
```
union { 成员列表 } 共用体对象名列表;
```

- 其中第一种形式最常用。

8.6.2 共用体对象的定义

- 定义共用体对象时可以进行初始化，但只能按一个成员给予初值，例如：

```
union A x={ 5678 }; //正确，只能给出1个初值
union A y={5,6,7,8}; //错误，试图给出4个初值（结构体做法）
```

8.6.3 共用体对象的使用

- 共用体对象的使用主要是引用它的成员，方法是对象成员引用运算（. ），例如：

```
x.m=5678; //给共用体成员赋值
printf("%d,%d,%d,%d\n",x.m,x.n,x.a,x.b);
//输出共用体成员 5678,5678,46,46
scanf("%d%d%d%d",&x.m,&x.n,&x.a,&x.b); //输入共用体成员
x.n++; //共用体成员运算
```

8.6.3 共用体对象的使用

- 程序中，第1句给成员m赋值5678，由于所有成员内存是共享的，因此每个成员都是这个值。
- 第2句输出m和n为5678，输出a和b为46，因为a和b类型为char，仅使用共享内存中的一部分（4个字节的低字节），即5678（0x162E）的0x2E（46）。
- 同时每个成员的起始地址是相同的，当运行第3句时输入1 2 3 4↙，x.m得到1，但紧接着x.n得到2时，x.m也改变为2了（因为共享），依次类推，最终x.b得到4时，所有成员都是这个值。第4句当x.n自增运算后，所有成员的值都改变了。

8.6.3 共用体对象的使用

- 由于成员是共享存储空间的，使用共用体对象成员时有如下特点：
- ①修改一个成员会使其他成员发生改变，所有成员存储的总是最后一次修改的结果；
- ②所有成员的值是相同的，区别是不同类型决定使用这个值的全部或部分；
- ③所有成员的起始地址值是相同的，因此通常只按一个成员输入、初始化；

8.6.3 共用体对象的使用

- 不能对共用体对象整体进行输入、输出、算术运算等操作，只能对它赋值操作。赋值实际上就是将一个对象的内容按内存形式完全复制到另一个对象中，例如：

```
union A one,two={1234};
one=1234; //错误，类型不兼容
one=two; //正确，赋值时复制two的内存数据到one中
```

8.6.4 结构体与共用体嵌套

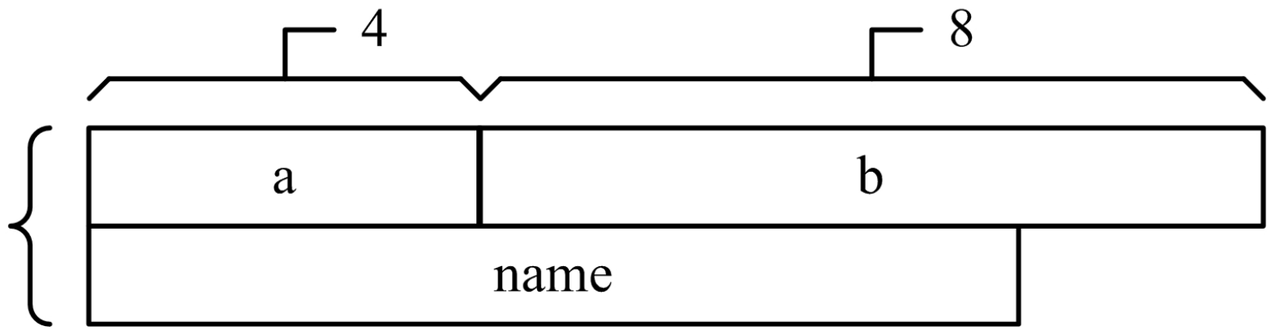
- 如何才能做到两组不同类型成员共享呢？方法是将其设计为结构体类型，再将这些结构体类型构造为共用体类型。例如：

```
struct tagDATA1 {
    int a; //整型成员
    double b; //浮点型成员
};
struct tagDATA2 {
    char name[10]; //字符串成员
};
union tagDATA12 {
    struct tagDATA1 a;
    struct tagDATA2 b;
};
```

8.6.4 结构体与共用体嵌套

- union tagDATA12内存形式如图所示。

图8.6 共用体嵌套结构体类型的内存结构



8.6.4 结构体与共用体嵌套

- 在共用体中嵌套结构体类型，可以解决复杂数据类型之间共享内存的需求。

- 在结构体中嵌套共用体类型，可以节省存储空间。


程序设计