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

- 3、复杂数据为元素的数组
- 4、用指针简洁表示复杂数据

8.3 结构体与数组

- 结构体的成员可以是数组，数组元素也可以是结构体类型。

8.3.1 结构体数组

- 数组元素可以是结构体类型，称为结构体数组，如一维结构体数组定义形式为：

```
struct 结构体类型名 结构体数组名[常量表达式];
```

8.3.1 结构体数组

- 例如表示平面上若干个点的数据对象，可以这样定义：

```
struct tagPOINT { //点类型
    int x,y; //平面上点的x、y坐标
};
struct tagPOINT points[100]; //表示100个点的数据对象
```

- 结构体数组的内存形式是按数组内存形式排列每个元素，每个元素按结构体内存形式排列。如points数组的内存长度是100*sizeof(struct tagPOINT)。

8.3.1 结构体数组

- 与其他数组一样，可以对结构体数组进行初始化，如一维结构体数组初始化形式为：

```
struct 结构体类型名 结构体数组名[常量表达式]={初值序列};
```

- 其中初值序列必须按内存形式做到类型、次序一一对应。

8.3.1 结构体数组

- 例如表示平面上3个矩形框的数据对象，可以这样定义：

```
struct tagRECT { //矩形框类型
    int left,top,right,bottom;
    //平面上矩形框左上角和右下角的x、y坐标
};
struct tagRECT rects[3]={{1,1,10,10},{5,5,25,32},
                         {100,100,105,200}};
```

8.3.1 结构体数组

- 初值写法中除最外面的一对大括号外，其他大括号可以省略。例如：

```
struct tagRECT _rect[3]={1,1,10,10,5,5,25,32,100,100,105,200};
```

- 在初始化时，数组元素个数可以不指定，而由编译器根据初值自动确定，例如：

```
struct tagRECT r[]={{1,1,10,10},{5,5,25,32},{100,100,105,200}};
```

8.3.1 结构体数组

- 引用结构体数组成员需要将数组下标运算、对象成员引用运算结合起来操作，其一般形式为：

```
数组对象[下标表达式].成员名
```

- 示例

```
r[0].left=r[0].top=10; //数组对象[下标表达式]是结构体对象
```

8.3.2 结构体数组成员

- 结构体类型中可以包含数组成员，数组成员类型既可以是基本数据类型又可以是指针类型、结构体类型，例如表示平面三角形的数据对象，可以这样定义：

```
struct tagTRIANGLE { //三角形类型
  struct tagPOINT p[3]; //由3个平面上的点描述三角形
};
```

8.3.2 结构体数组成员

- 引用结构体数组成员需要将对象成员引用运算、数组下标运算结合起来操作，其一般形式为：

```
结构体对象.数组成员[下标表达式]
```

- 示例

```
struct tagTRIANGLE tri;
tri.p[0].x=10,tri.p[0].y=10;//结构体对象.数组成员[下标表达式].成员名
```

8.3.2 结构体数组成员

【例8.1】

从键盘上输入20个学生信息记录（包含学号、姓名、成绩），按成绩递减排序；当成绩相同时，按学号递增排序。

8.3.2 结构体数组成员

例8.1

```
1  #include <stdio.h>
2  #define N 20
3  struct tagSTUDENT { //学生信息类型
4    int no; //学号
5    char name[21]; //姓名
6    double score; //成绩
7  };
8  int main()
9  {
10   struct tagSTUDENT A[N] , t;
11   int i , j=2.0; //消除浮点型bug
12   for (i=0; i<N; i++) //输入学生信息
13     scanf("%d%s%lf",&A[i].no,A[i].name,&A[i].score);
14   for (i=0; i<N-1; i++) //排序
15   for (j=i; j<N; j++)
```

8.3.2 结构体数组成员

例8.1

```
16      if (A[i].score<A[j].score //按成绩递减排序
17      || (A[i].score==A[j].score&&A[i].no>A[j].no)) //按学号递增排序
18          t=A[i], A[i]=A[j], A[j]=t;
19    for (i=0; i<N; i++) //输出学生信息
20      printf("%d,%s,%f\n",A[i].no,A[i].name,A[i].score);
21    return 0;
22 }
```

8.4 结构体与指针

- 可以定义指向结构体类型的指针，也可以定义指向结构体数组的指针，结构体成员还可以是指针类型。

8.4.1 指向结构体的指针

- 可以得到结构体对象各成员的地址，方法是取地址运算（&）（或数组名即是地址），指向成员的指针类型应与成员类型一致。

8.4.1 指向结构体的指针

- 示例

```
struct tagSTAFF m; //结构体对象
int *p1; //指向no成员的指针类型是int*
char *s1,*s2; //指向name、sex成员的指针类型是char*
struct tagDATE *p2; //指向birthday成员的指针类型是struct tagDATE*
p1=&m.no; //取no成员的地址
s1=m.name; //name成员是数组，数组名即是地址
s2=&m.sex; //取sex成员的地址
p2=&m.birthday; //取birthday成员的地址
```

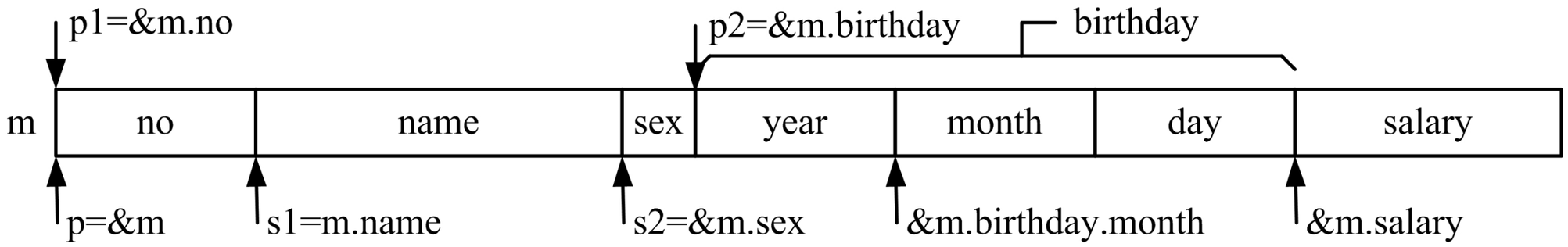
8.4.1 指向结构体的指针

- 也可以得到结构体对象的地址，方法是取地址运算（&），指向结构体对象的指针类型必须是结构体类型。例如：

```
struct tagSTAFF m, *p; //指向结构体对象的指针
p=&m; //取结构体对象的地址
```

8.4.1 指向结构体的指针

- 指向结构体对象成员的指针值是该成员内存单元的起始地址，指向结构体对象的指针值是该对象内存单元的起始地址。显然，结构体对象的地址值（&m）与第一个成员的地址值（&m.no）相同。如图所示显示指向结构体成员和指向结构体对象的指针。



8.4.1 指向结构体的指针

- 假设p是指向结构体对象的指针，通过p引用结构体对象成员有两种方式：
- ①对象法：(*p)．成员名；
- ②指针法：p->成员名。

8.4.1 指向结构体的指针

表8-2 指针成员引用运算符

运算符	功能	目	结合性	用法
->	指针成员引用运算	双目	自左向右	pointer->member

指针成员引用运算符在所有运算符中优先级较高，其作用是通过结构体指针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。

8.4.1 指向结构体的指针

- 示例

```
p->no=10002;
//将10002赋值给对象中的no成员，指针成员引用运算结果是左值（成员本身）
p->salary=p->salary+500.0; //在表达式中引用指针指向的成员
p->no++; //按优先级等价于(p->no)++
```

8.4.1 指向结构体的指针

- 指针成员引用运算时需要注意以下几点：
- （1）指针成员引用运算符（->）左边运算对象pointer必须是指向结构体对象的指针，可以是变量或表达式，右边member必须是结构体对象中的成员名。其引用形式为：

```
结构体指针->成员名
```

8.4.1 指向结构体的指针

- （2）如果成员本身又是一个结构体对象指针，就要用指针成员引用运算符一级一级地引用成员。例如：

```
struct tagDATE d={1981,1,1};
struct tagTEACHER { //教师信息类型
    int no; //工号
    char name[21]; //姓名
    struct tagDATE *pbirthday; //出生日期
} a={1001,"Li Min",&d}, *p=&a;
```

8.4.1 指向结构体的指针

- 结构体对象a初始化时，其成员pbirthday是struct tagDATE*指针，指向结构体对象d。结构体指针p指向结构体对象a。则

```
p->no=10001; //通过指针p引用a的no成员
p->pbirthday->year=2008;//通过指针p->pbirthday引用d的year成员
```

8.4.2 指向结构体数组的指针

- 指向结构体数组的指针本质上是数组指针，其定义的一般形式为：

```
struct 结构体类型名 结构体数组名[常量表达式],*结构体指针;

结构体指针=结构体数组名; //指向结构体数组
```

8.4.2 指向结构体数组的指针

- 假设指向结构体数组的指针p指向了结构体数组首地址，通过p可以按下面的形式访问第i个数组元素（结构体对象）：
- ①p[i]：数组法访问数组元素；
- ②*(p+i)：指针法访问数组元素。

8.4.2 指向结构体数组的指针

- 通过p还可以按下面的形式访问结构体数组成员：
- ①p[i]．成员名：结合数组法与对象法访问结构体数组成员。
- ②(*(p+i))．成员名：结合指针法与对象法访问结构体数组成员，注意（*）的优先级低于（．）。
- ③(p+i)->成员名：指针法访问结构体数组成员。

8.4.2 指向结构体数组的指针

- 示例

```
int i=3,j=5;
struct tagSTAFF branch[20], *p=branch; //指向结构体数组的指针
p[i]=p[j]; //等价于branch[i]=branch[j]，结构体对象赋值
*p=*(p+1); //等价于branch[0]=branch[1]，结构体对象赋值
p[i].no=10003; //给结构体对象成员赋值
printf("%d\n",(*(p+i)).no); //输出结构体对象成员
scanf("%s",(p+i)->name); //输入结构体对象成员
```

8.4.3 结构体指针成员

- 结构体成员可以是指针类型，例如：

```
struct tagDATA1 {
    int data; //整型成员
    char *name; //指针成员
} a={10,"Li Min"} , b;
```

8.4.3 结构体指针成员

- 结构体对象的指针成员存储的是地址，而不是所指向的内容。如a.name的值是字符串常量“Li Min”的地址，而不是字符串本身，这点与数组成员是不同的，例如：

```
struct tagDATA2 {
    int data; //整型成员
    char name[10]; //数组成员
} c={10,"Li Min"} , d;
```

- 结构体对象c存储字符串"Li Min"。

8.4.3 结构体指针成员

- 上述两种结构体类型的内存长度是不一样的：

```
sizeof(struct tagDATA1) //长度为8
sizeof(struct tagDATA2) //长度为14
```

```
struct tagDATA1 {
    int data; //整型成员
    char *name; //指针成员
} a={10,"Li Min"} , b;
```

```
struct tagDATA2 {
    int data; //整型成员
    char name[10]; //数组成员
} c={10,"Li Min"} , d;
```

8.4.3 结构体指针成员

- 当这两种结构体对象赋值时，含义也是不同的，例如：

```
b=a;//复制一个整型和一个指向"Li Min"的指针，b中name指向"Li Min"
d=c;//复制一个整型和"Li Min"，d中name为"Li Min"
```

```
struct tagDATA1 {
    int data; //整型成员
    char *name; //指针成员
} a={10,"Li Min"} , b;
```

```
struct tagDATA2 {
    int data; //整型成员
    char name[10]; //数组成员
} c={10,"Li Min"} , d;
```

8.4.3 结构体指针成员

- 结构体指针成员可以指向结构体类型，甚至是自身类型。例如：

```
struct tagNODE {
    int data; //整型成员
    struct tagNODE *next; //指针成员，指向自身类型的指针
} a,b,c;
```

- 其中next成员指向struct tagNODE。

8.4.3 结构体指针成员

- 假设

```
a.next=&b; //对象a的next指向对象b
b.next=&c; //对象b的next指向对象c
```

- 那么访问a、b、c三个对象的data成员可以有如下写法：

```
a.data=1; //访问对象a的data成员
a.next->data=2; //访问对象b的data成员，等价于b.data
a.next->next->data=3; //访问对象c的data成员，等价于c.data
```

8.4.3 结构体指针成员

- 即通过next成员可以将三个对象“链接”起来，形成链表结构。

- 在这种结构中，只要知道第一个对象（如a），不需要知道其他对象（如b、c），就可以访问所有在链表上的对象。如果链接的对象是用动态内存分配得到的，则对象只有地址没有名称，那么这样的指针访问就显得很有意义了。


程序设计