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

- 3、枚举数据类型
- 4、位域数据类型

8.7 枚举类型

- 枚举类型是C语言的一种构造类型。它用于声明一组命名的常数，当一个变量有几种可能的取值时，可以将它定义为枚举类型。

8.7.1 枚举类型的声明

- 枚举类型是由用户自定义的由多个命名枚举常量构成的类型，其声明形式为：

```
enum 枚举类型名 {命名枚举常量列表};
```

- enumerate，枚举，列举

8.7.1 枚举类型的声明

- 示例

```
enum tagDAYS {MON,TUE,WED,THU,FRI,SAT,SUN};
```

- enum tagDAYS是枚举类型，MON等是命名枚举常量。默认时枚举常量总是从0开始，后续的枚举常量总是前一个的枚举常量加一。如MON为0，TUE为1，……，SUN为6。

8.7.1 枚举类型的声明

- 可以在（仅仅在）声明枚举类型时，为命名枚举常量指定值。例如：

```
enum tagCOLORS {RED=10,GREEN=8,BLUE,BLACK,WHITE};
```

- 则RED为10、GREEN为8、BLUE为9、BLACK为10、WHITE为11。

8.7.1 枚举类型的声明

- 命名枚举常量是一个整型常量值，也称为枚举器（enumerator），在枚举类型范围内必须是唯一的。命名枚举常量是右值不是左值，例如：

```
RED=10; //错误，RED不是左值，不能被赋值
GREEN++; //错误，GREEN不是左值，不能自增自减
```

8.7.2 枚举类型对象

- 定义枚举类型对象有三种形式：

```
①enum 枚举类型名 {命名枚举量列表} 枚举对象名列表;

②enum 枚举类型名 枚举对象名列表;
//在已有枚举类型下，最常用的定义形式

③enum {命名枚举量列表} 枚举对象名列表;
//使用较少的定义形式
```

8.7.2 枚举类型对象

- 可以在定义对象时进行初始化，其形式为：

```
枚举对象名1=初值1，枚举对象名2=初值2，.....;
```

- 例如：

```
enum tagDIRECTION{LEFT,UP,RIGHT,DOWN,BEFORE,BACK} dir=LEFT;
```

8.7.2 枚举类型对象

- 本质上，枚举类型对象是其值限定在枚举值范围内的整型变量。

- 在许多应用程序中，例如设计使用操作杆的游戏程序，代表操作方向的变量的取值就希望是有限集合常量，这时使用枚举类型很方便。

8.7.2 枚举类型对象

- 当给枚举类型对象赋值时，若是除枚举值之外的其他值，编译器会给出错误信息，这样就能在编译阶段帮助程序员发现潜在的取值超出规定范围的错误。例如：

```
enum tagCOLORS color;
color=101; //错误，不能类型转换
color=(enum tagCOLORS)101; //正确，但结果没有定义
```

8.8 位域

- “位域”是把一个字节中的二进位划分为几个不同的区域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作。这样就可以把几个不同的对象用一个字节的二进制位域来表示。

8.8.1 位域的声明

- 在声明结构体、共用体类型时，可以指定其成员占用存储空间的二进制位数，这样的成员称为位域（bit field），又称位段。其声明形式为：

```
struct/union 结构体/共用体类型名 {
   位域类型 成员名:常量表达式; //声明位域
   成员类型 成员名; //数据成员列表
};
```

- 其中常量表达式为非负整数值，用来指明位域所占存储空间的二进制位长度；位域类型必须是unsigned int或int，有的编译器如VC还允许char和long及其unsigned类型。

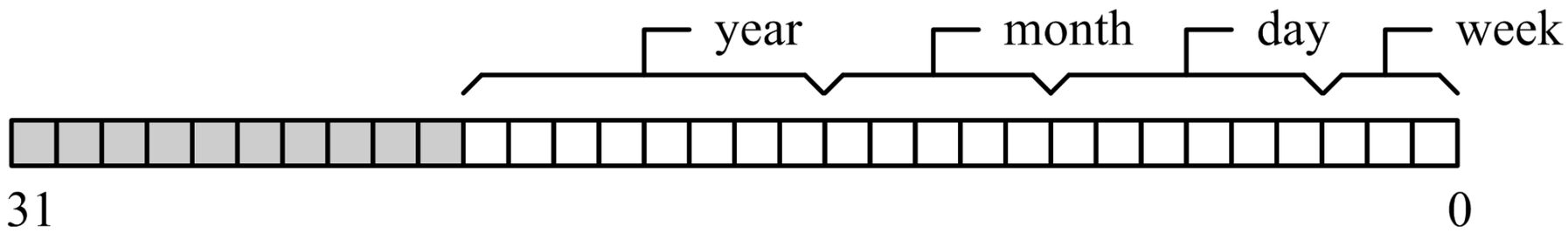
8.8.1 位域的声明

- 示例

```
struct tagDATE { //日期类型
    int week  :3; //3位，星期
    int day   :6; //6位，日
    int month :5; //5位，月
    int year  :8; //8位，年
};
```

8.8.1 位域的声明

- 数据的内存形式如下：



- 成员week、day、month、year分别占3、6、5、8位。在存储位域时，一般从存储单元的低位至高位分配位域，具体因编译器而异，使用位域时可以不关心这个细节。

8.8.1 位域的声明

- 本质上，位域是按其类型所对应的存储单元（如int）存放的，即将位域存放在一个单元（int）内，若位数不够时再分配一个单元（int），直至能够容纳所有的位域。如上述位域（均为int）共22位，因此需要一个int（32位，4个字节）来存储。

8.8.1 位域的声明

- 示例

```
struct tagBITDATA { //总计4+1+2+1个字节，32+8+16+8位
  int a:1;//分配1个int（4字节，32位）存储a，只用其中的1位，其余空闲不用。
  char b:2;
  //分配1个char（1字节，8位）存储b，只用其中的2位，其余空闲不用。
  char c:2;//使用前面分配的char存储c（归并分配），用其中的2位。
  short d:3;
  //分配1个short（2字节，16位）存储c，只用其中的3位，其余空闲不用。
  unsigned char data;
  //数据成员（非位域成员），unsigned char型（1字节，8位）
};
```

8.8.1 位域的声明

- 一般地，设m个类型位域的二进制位长度为 $L_i(L_i \geq 1), i = 1, 2, \cdots, m$，位域类型对应的存储单元大小为 Z_i（字节），则分配得到的存储单元数 $n = \sum([(L_i - 1) / (Z_i * 8)] + 1) * Z_i$（方括号表示取整）。而非位域成员由其类型决定长度。如struct tagBITDATA的存储单元数 为：

$$n = ([(1-1)/(4*8)]+1)*4 + ([(4-1)/(1*8)]+1)*1 + ([(1-1)/(2*8)]+1)*2 + 1 = 8$$

8.8.1 位域的声明

- 若存储单元位长度大于位域总长度，则多余二进制位空闲不用。包含位域的结构体和共用体同样有字节对齐的问题。

8.8.1 位域的声明

- 位域的长度不能超过其类型对应的存储单元的大小，且必须存储在同一个存储单元中，不能跨两个单元。如果一个单元剩余存储空间不能容纳下一个位域，则该空间闲置不用，直接从下一个单元起存储位域。

8.8.1 位域的声明

- 示例

```
struct tagBITDATA3 {
    int  x:33; //错误,指定的位数超过int的容量
    char y:9;  //错误,指定的位数超过char的容量
    unsigned char m:6; //6位，剩余2位空闲不用
    unsigned char n:7; //3位，剩余1位空闲不用
};
```

8.8.1 位域的声明

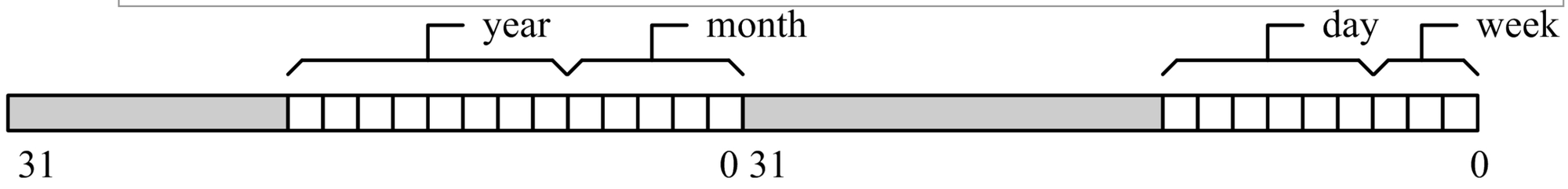
- 位域可以是匿名成员，即只指定位长度而不声明成员名称，例如：

```
struct tagBITDATA2 { //4个字节（unsigned）
    unsigned a:1;   //1位
    unsigned  :2;   //匿名位域成员，2位，但空闲不用
    unsigned c:3;   //3位，剩余26位空闲不用
};
```

8.8.1 位域的声明

- 匿名位域占用二进制位，但空闲不用。若匿名位域的位长度为0，则表示其后的位域成员将另起一个存储单元。例如：

```
struct tagDATE { //2个int，8个字节
    int week  :3; //3位，星期
    int day   :6; //6位，日
    int       :0; //强制后面的位域成员从一个新的int开始存储
    int month :5; //5位，月
    int year  :8; //8位，年
};
```



8.8.1 位域的声明

- 不能声明数组形式和指针形式的位域。位域只能是结构体、共用体类型的一部分，而不能单独定义。

- 位域适合系统编程、硬件编程，广泛地应用于操作系统、设备驱动、检测与控制、嵌入式控制系统等领域。之所以使用位域成员，原因有二：一是这些领域中某些数据往往对应硬件的物理信号线，决定了数据是二进制位形式而不是字节形式。二是以硬件为基础的系统编程中，存储容量是有限的，尽量节省存储空间是这些领域编程时的重要原则。

8.8.2 位域的使用

- 包含位域对象的定义形式与结构体对象的定义形式完全相同，例如：

```
struct tagC8253 { //8253定时器数据类型
    unsigned char  CLK:3; //8253时钟输入线
    unsigned char GATE:3; //8253门控信号
    unsigned char  OUT:3; //输出信号
    unsigned char    A:2; //地址编码
    unsigned char   CS:1; //片选信号
    unsigned char   RD:1; //控制器读信号
    unsigned char   WR:1; //控制器写信号
} m,n={0,1,0,2,1,1,0}; //初始化
```

8.8.2 位域的使用

- 其使用与结构体对象相同，如整体对象可以初始化、赋值，但不能做算术运算、输入输出。
- 通过成员引用运算符（. ）来使用位域，例如：

```
n.GATE=2; //位域赋值
n.CS=m.GATE >>1 && 0x3; //位域运算
printf("%d,%u,%x,%o\n",m.OUT,m.GATE,m.A,m.CLK); //输出位域
```

8.8.2 位域的使用

- 运算时，位域自动转换成整型。由于位域只有部分的二进制位，实际编程中需要注意其数值范围。如m.A只有2位，因此有效的数值只能是0～3，超出这个范围的二进制位会自动被截掉。如给m.A赋值8，m.A得值0。

8.8.2 位域的使用

- 由于位域是内存单元里面的一段，因而没有地址，故不能对位域取地址，如

```
scanf("%d%d",&m.GATE,&m.A); //错误，位域没有地址
```

- 也不能定义指针指向位域或函数返回位域。


程序设计