有些信息在存储时，并不需要占用一个完整的字节， 而只需占几个或一个二进制位。例如在存放一个开关量时，只有0和1 两种状态， 用一位二进位即可。为了节省存储空间，并使处理简便，Ｃ语言又提供了一种数据结构，称为“位域”或“位段”。所谓“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。一、位域的定义和位域变量的说明位域定义与结构定义相仿，其形式为：   
struct 位域结构名   
{ 位域列表 };  
其中位域列表的形式为： 类型说明符 位域名：位域长度   
例如：   
struct bs  
{  
int a:8;  
int b:2;  
int c:6;  
};位域变量的说明与结构变量说明的方式相同。 可采用先定义后说明，同时定义说明或者直接说明这三种方式。例如：   
struct bs  
{  
int a:8;  
int b:2;  
int c:6;  
}data;说明data为bs变量，共占两个字节。其中位域a占8位，位域b占2位，位域c占6位。对于位域的定义尚有以下几点说明：  
  
1. 一个位域必须存储在同一个字节中，不能跨两个字节。如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。例如：   
struct bs  
{  
unsigned a:4  
unsigned :0 /\*空域\*/  
unsigned b:4 /\*从下一单元开始存放\*/  
unsigned c:4  
}  
在这个位域定义中，a占第一字节的4位，后4位填0表示不使用，b从第二字节开始，占用4位，c占用4位。  
  
2. 由于位域不允许跨两个字节，因此位域的长度不能大于一个字节的长度，也就是说不能超过8位二进位。  
  
3. 位域可以无位域名，这时它只用来作填充或调整位置。无名的位域是不能使用的。例如：   
struct k  
{  
int a:1  
int :2 /\*该2位不能使用\*/  
int b:3  
int c:2  
};从以上分析可以看出，位域在本质上就是一种结构类型， 不过其成员是按二进位分配的。  
  
二、位域的使用位域的使用和结构成员的使用相同，其一般形式为： 位域变量名·位域名 位域允许用各种格式输出。  
main(){  
struct bs  
{  
unsigned a:1;  
unsigned b:3;  
unsigned c:4;  
} bit,\*pbit;  
bit.a=1;  
bit.b=7;  
bit.c=15;  
printf("%d,%d,%d\n",bit.a,bit.b,bit.c);  
pbit=&bit;  
pbit->a=0;  
pbit->b&=3;  
pbit->c|=1;  
printf("%d,%d,%d\n",pbit->a,pbit->b,pbit->c);  
} 上例程序中定义了位域结构bs，三个位域为a,b,c。说明了bs类型的变量bit和指向bs类型的指针变量pbit。这表示位域也是可以使用指针的。  
程序的9、10、11三行分别给三个位域赋值。( 应注意赋值不能超过该位域的允许范围)程序第12行以整型量格式输出三个域的内容。第13行把位域变量bit的地址送给指针变量pbit。第14行用指针方式给位域a重新赋值，赋为0。第15行使用了复合的位运算符"&="， 该行相当于： pbit->b=pbit->b&3位域b中原有值为7，与3作按位与运算的结果为3(111&011=011,十进制值为3)。同样，程序第16行中使用了复合位运算"|="， 相当于： pbit->c=pbit->c|1其结果为15。程序第17行用指针方式输出了这三个域的值。