

C语言与程序设计

The C Programming Language




第9章 结构与联合

毛伏兵

华中科技大学计算机学院

9.1 结构与联合

结构与联合都属于C的构造类型。

- 
- 某种具体的结构说明（关键字**struct**）
 - 结构变量的定义
 - 结构成员的引用（.运算符）
 - 通过结构指针引用结构成员（->运算符）
 - 字段结构
 - 联合类型（**union**）

例1、输入平面上两个点，求它们之间的长度。

```
#include "stdio.h"
#include "math.h"
struct point{ /* 平面上点的结构类型说明 */
    int    x;
    int    y; /* x,y是点的坐标 */
};
int main(void)
{ struct point start, end; /* 声明点结构变量 */
  double dx,dy,length;
  printf("输入线段的起点和终点坐标: \n");
  scanf("%d%d%d%d",&start.x,&start.y, &end.x,&end.y);
  dx=(end.x-start.x)*(end.x-start.x);
  dy=(end.y-start.y)*(end.y-start.y);
  length=sqrt(dx+dy); /*计算线段长度*/
  printf("the length is %f\n",length); /*输出线段长度*/
  return 0;
}
```

结构成员的访问：
结构变量名. 成员名

例2、改写例1将求两点间的长度定义为函数

```
double  linelen(struct point  pt1,struct point  pt2)
{
    double dx,dy,length;
    dx=(pt2.x-pt1.x)*(pt2.x-pt1.x);
    dy=(pt2.y-pt1.y)*(pt2.y-pt1.y);
    length=sqrt(dx+dy);          /*计算线段长度*/
    return length;
}
```

结构变量作为函数的参数

```
#include "stdio.h"
#include "math.h"
struct point{ /* 平面上点的结构类型 */
    int    x;
    int    y; /* x,y是点的坐标 */
};
int main(void)
{ struct point start, end; /* 声明点结构变量 */
  double length;
  printf("输入线段的起点和终点坐标: \n");
  scanf("%d%d%d%d",&start.x,&start.y, &end.x,&end.y);
  length=linelen(start, end); /*计算线段长度*/
  printf("the length is %f\n",length); /*输出线段长度*/
  return 0;
}
```

例3、改写例2将函数参数改为结构类型的指针

```
double  linelen(struct point *p1,struct point *p2)
{
    double dx,dy,length;
    dx=(p2->x-p1->x)*(p2->x-p1->x);
    dy=(p2->y-p1->y)*(p2->y-p1->y);
    length=sqrt(dx+dy); /*计算线段长度*/
    return length;
}
```

- 结构类型的指针作为函数的参数
- 结构类型的指针也可以作为函数的返回值

```
#include "stdio.h"
#include "math.h"
struct point{ /* 平面上点的结构类型 */
    int    x;
    int    y;    /* x,y是点的坐标 */
};
int main(void)
{ struct point start, end; /* 声明点结构变量 */
  double length;
  printf("输入线段的起点和终点坐标: \n");
  scanf("%d%d%d%d",&start.x,&start.y, &end.x,&end.y);
  length=linelen(&start, &end); /*计算线段长度*/
  printf("the length is %f\n",length); /*输出线段长度*/
  return 0;
}
```

结构类型的变量作为函数的参数时：



**1、将实参结构拷贝给形参，占用内存较多，
耗费时间较长。**

—— 适用于较小的结构

2、值传递，形参结构不影响实参结构的值。

结构类型的指针作为函数的参数时:

1、只将实参指针的值拷贝到形参指针单元。
占用内存少，耗费时间短。

—— 适用于较大的结构

2、在函数内部对形参指针的间访操作会影响
实参指针所指结构变量的值。

—— 适合于要修改实参指针所
指的结构变量值的情况。

通过结构指针访问结构成员

- `struct point a, b={10, 20}, *p=&a;`
 `*p=b;` `/* 与a=b;等价 */`
- 通过“*”用结构指针访问结构变量的成员
 $(\text{*结构指针}).\text{成员名}$
 $(\text{*p}).x \iff a.x$
- 通过运算符“->”访问结构变量的成员
 $\text{结构指针名} \rightarrow \text{结构成员名}$
 $p \rightarrow y \iff (\text{*p}).y$

结构变量的赋值操作

- 当两个结构变量的类型相同时，它们之间可以直接相互赋值。例如：

```
static struct point {
```

```
    int x;
```

```
    int y;
```

```
} a={1,2}, b;
```

则 **b=a;** /* 合法,对应的各个成员赋值 */

b={1,2}; 对的

/* 错

***/??**

定义一个初始化点结构变量的函数

- 结构成员作为函数的参数
- 结构变量作为函数的返回值

```
point makepoint(int x, int y)
{
    struct point temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

```

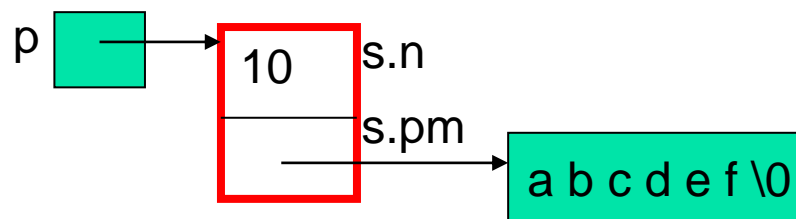
#include "stdio.h"
#include "math.h"
struct point{ /* 平面上点的结构类型 */
    int    x;
    int    y;    /* x,y是点的坐标 */
};
int main(void)
{ struct point start, end; /* 声明点结构变量 */
  double length;
  start=makepoint (0, 0) ;
  end=makepoint (2, 3) ;
  length=linelen(&start, &end); /*计算线段长度*/
  printf("the length is %f\n",length); /*输出线段长度*/
  return 0;
}

```

设有：

```
struct T{
    int n;
    char *pm;
```

}s={10, “abcdef ” },*p=&s; 写出以下表达式的值和类型。
(各表达式相互无关)



- 1) ++p->n
- 2) p->n++
- 3) *p->pm
- 4) *p->pm++
- 5) *++p->pm

表达式	值与类型	表达式执行的操作	等价的表达式
++p->n	11, int	访问 n 并使其增 1	++s.n 和 ++(*p).n
p->n++	10, int	访问 n, 取其原值参与运算, 再使 n 增 1	s.n++ 和 (*p).n++
*p->pm	'a', char	访问 pm 所接字符 'a'	*s.pm 和 *(*p).pm
*p->pm++	'a', char	访问 pm 所接字符 'a' 后 pm 增 1 指向 'b'	*s.pm++ 和 *(*p).pm++
*++p->pm	'b', char	先访问 pm, 然后 pm 增 1, 再访问 pm 所接字符 'b'	++s.pm 和 ++(*p).pm

所有运算符的优先级和结合性规则

优先级 (从高到低)	运算符	运算符名称	结合性
1	() [] > . ~	圆括号 下标 间接引用结构体成员	左结合
单目运算符	!	逻辑非	
	~	按位取反	
	++、--	自增、自减	
2	+、-	取正、取负	右结合
	(数据类型) &、* sizeof	强制类型转换 取地址、间接引用 数据长度	
3	*/、/、%	乘、除、求余数	左结合
4	+、-	加、减	左结合
5	<<、>>	左移、右移	左结合
6	<、> >=、<=	大于、小于 大于等于、小于等于	左结合
7	=、!=	等于、不等于	左结合
8	&	按位与	左结合
9	^	按位异或	左结合
10		按位或	左结合
11	&&	逻辑与	左结合
12		逻辑或	左结合
13	? :	条件	右结合
14	=、+=、-=、*=、/=、%=、>>=、 <<=、&=、^=、 =	赋值	右结合
15	,	逗号	左结合

- 例4 编程输入商品信息（包括商品编码、名称、价格），并且按照价格排序并显示排序后的结果。

商品编码	名称	价格（元）
1	笔	2.0
2	毛巾	10.5


```
#include<stdio.h>
#define N 3
struct GOODS {
    long code;        /* 货物编码 */
    char name[20];    /* 名称 */
    float price;      /* 价格 */
};
/* 输入n件物品的信息 */
void input(struct GOODS *p,int n);
/* 对n件物品按价格降序排序 */
void sort(struct GOODS *p,int n);
/*显示n件物品的信息 */
void display(struct GOODS *p,int n);
```

```
int main(void)
```

```
{
```

```
    struct GOODS g[N];
```

```
    input(g,N);    /* 结构数组名作为实参 */
```

```
    display(&g[0],N);
```

```
    sort (g,N);
```

```
    display(g,N);
```

```
    return 0;
```

```
}
```

/* 输入n件物品的信息 */

void input(struct GOODS *p,int n)

{

int i;

for(i=0;i<n;i++){

scanf("%ld",&p[i].code);

scanf("%s",(p+i)->name);

scanf("%f",&(p+i)->price;

}

}

/*显示n件物品的信息 */

void display(struct GOODS *p,int n)

{

int i;

for(i=0;i<n;i++){

printf("%ld\t",(*(p+i)).code);

printf("%s\t",(p+i)->name);

printf("%f\n",p[i].price);

}

}

/* 对n件物品按价格降序排序 */

void sort(struct GOODS *p,int n)

{ int i,j; struct GOODS t;

for(i=0;i<n-1;i++)

for(j=i+1;j<n;j++)

if((p+i)->price<(p+j)->price){

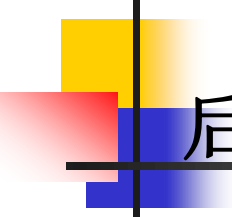
t=*(p+i);

***(p+i)=*(p+j);**

***(p+j)=t;**

}

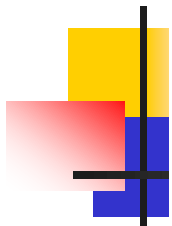
}



设计一个能够描述学生成绩的结构类型，然后声明对应的结构数组，描述以下成绩表。

成绩表

学号	姓名	性别	入学时间			计算机原理	英语	数学	音乐
			年	月	日				



定义

```
struct date
```

```
{
```

```
    int month;
```

```
    int da
```

```
    int year;
```

```
};
```

```
struct STUDENT
```

```
{
```

```
    int ID;
```

```
    char Name[10];
```

```
    char Sex[4];
```

```
    struct date timeOfEnter;
```

```
    int Computer;
```

```
    int English;
```

```
    int Math;
```

```
    int Music;
```

```
};
```

```
struct STUDENT stu[30];
```



初始化

```
struct STUDENT stu[30] = {  
    {1, " Tom  ", 'M', {1999, 12, 20}, 90, 83, 72, 82},  
    {2, " Nick  ", 'M', {1999, 07, 06}, 78, 92, 88, 78},  
    {3, " Sue   ", 'F', {1999, 07, 06}, 89, 72, 98, 66},  
    {4, " May   ", 'M', {1999, 07, 06}, 78, 95, 87, 90}  
};
```




嵌套结构中结构成员的访问

- **结构变量名.结构成员名.成员名**

- **理解为:**

(结构变量名.结构成员名) .成员名

stu[0].timeofenter.year

- 注意 “.” 、 “->” 、 “*” 、 “++或--” 运算符 优先级和结合性

- 设有说明：

```
char u[]="abcde";
```

```
char v[]="xyz";
```

```
struct T{
```

```
    int x;
```

```
    char c;
```

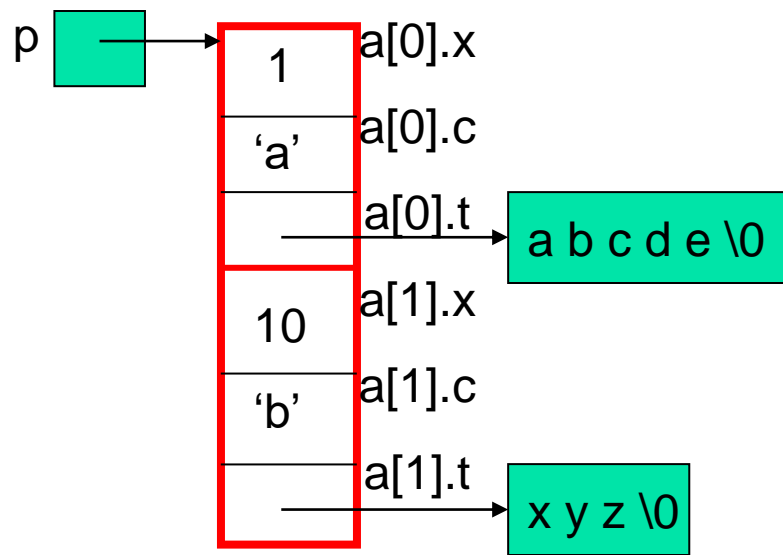
```
    char *t;
```

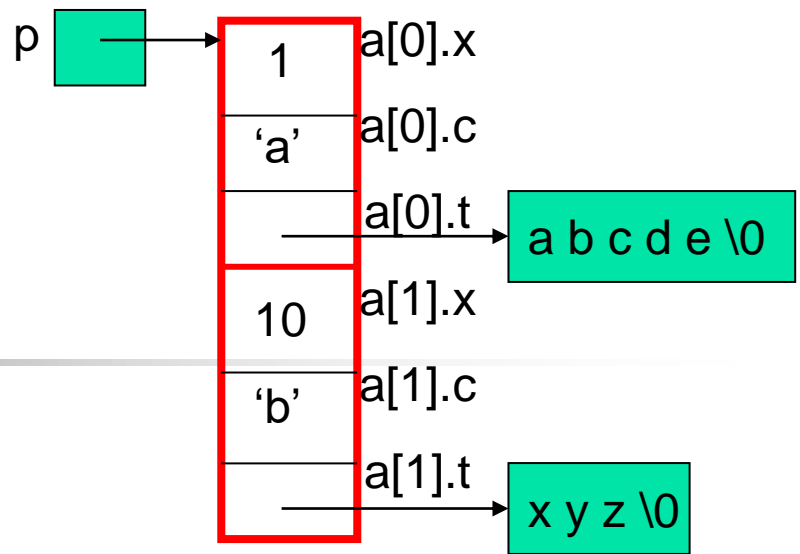
```
}a[]={ {1, 'a', u}, {10, 'b', v}}, *p=a;
```

- 若执行代码片段：

```
printf( "%d\n" ,(++p)->x);    /* 输出 10 */
```

```
printf( "%c\n",p->c);        /* 输出 b */
```





■ 若执行代码片段:

```
p=a;
```

```
printf("%d\n",(p++)->x); /* 输出 1 */
```

```
printf("%d\n",p->x); /* 输出 10 */
```

若执行代码片段:

```
p=a;
```

```
printf("%c\n",*p++->t); /* 输出 a
```

*/??Sequence?

```
printf("%c\n",*p->t); /* 输出 x */
```

比较下面各表达式的异同:

- $(++p) \rightarrow x$ 与 $++p \rightarrow x$ /* 不同 */ 优先级?
- $(p++) \rightarrow x$ 与 $p++ \rightarrow x$ /* 相同 */
- $*(p++) \rightarrow t$ 与 $*p++ \rightarrow t$ /* 相同 */
- $*(++p) \rightarrow t$ 与 $*++p \rightarrow t$ /* 不同 */
- $*(++p \rightarrow t)$ 与 $*++p \rightarrow t$ /* 相同 */
- $*++p \rightarrow t$ 与 $++*p \rightarrow t$ /* 不同 */



用sizeof计算数组元素的个数

设有数组a，表达式：

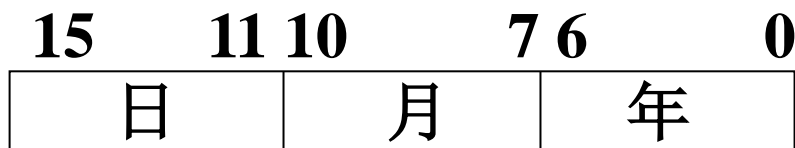
sizeof a/sizeof a[0]

计算a数组中元素的个数。

*9.8 字段 (bit field) 结构

【例2.16】 压缩和解压。把表示21世纪日期的日、月和年3个整数压缩成1个16位的整数。

分析： 因为日有31个值，月有12个值，年有100个值，所以可以在一个整数中用5b表示日，用4b表示月，用7b表示年。



用字段结构实现

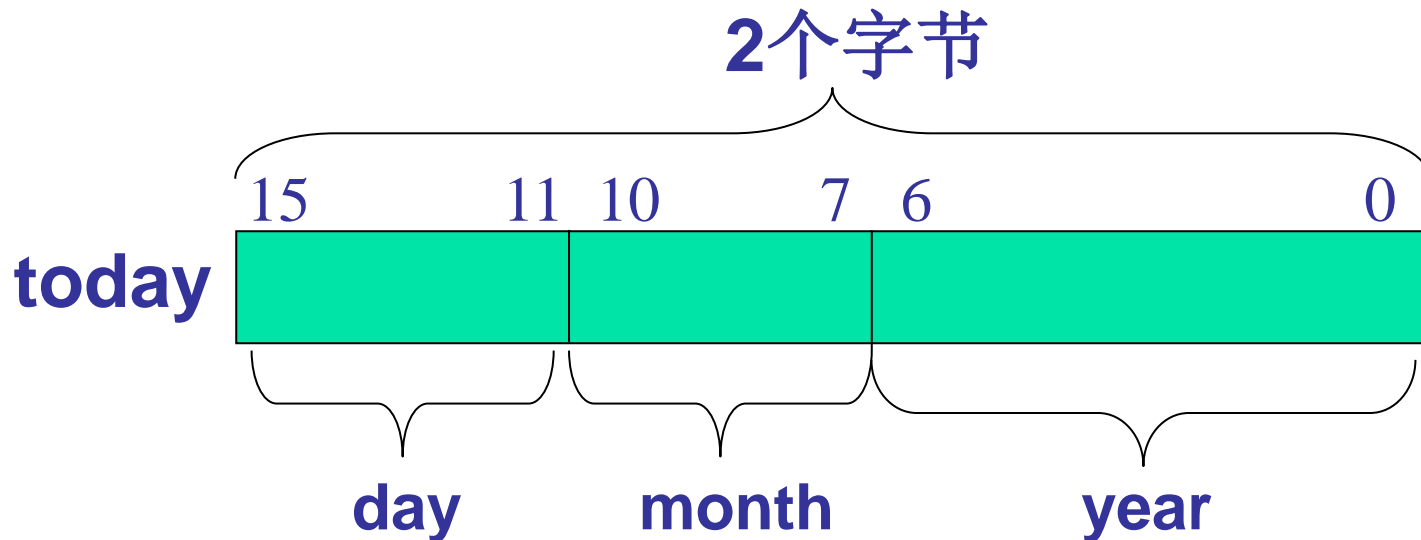
字段结构的声明:

```
struct date {  
    unsigned short year : 7 ;    /* 年 */  
    unsigned short month : 4 ;   /* 月 */  
    unsigned short day : 5 ;     /* 日 */  
};
```

↑ ↑
字段名 字段宽度

字段的存储结构

```
struct date {  
    unsigned short year : 7 ;    /* 年 */  
    unsigned short month : 4 ;   /* 月 */  
    unsigned short day : 5 ;     /* 日 */  
};  
struct date today; /* today是date 字段结构变量 */
```



引用字段

- 与结构完全相同： . 或 ->
- 字段就是一个小整数，它可以出现在其它整数可以出现的任何地方。字段在参与运算时被自动转换为 int 或 unsigned int 类型的整数。

```
today. year=2013;  
today. month=5;  
today. day=9;
```

字段：字中一组相邻的二进制位，是指定了存储位数的，**unsigned int**（或**unsigned short**）的结构成员。

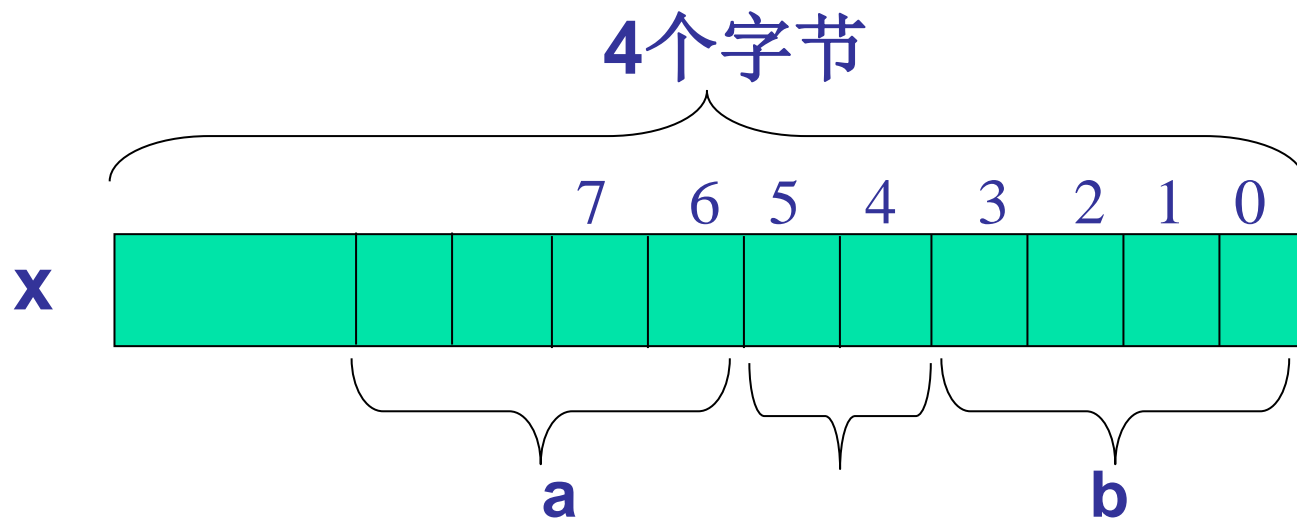
字段的宽度：组成字段的二进制位的数目，是一个非负的整型常量表达式。

字段结构在操作系统，编译程序，计算机接口的C语言编程方面使用较多。

可以用一个字段结构变量表示多个值很小的整数，通过字段名对相应的小整数进行操作。

无名位段（匿名）

```
struct {  
    unsigned a: 4;  
    : 2;  
    unsigned b: 4;  
} x;
```



9.7 联合

- **结构变量**是占据各自不同空间的各成员变量的集合。
- **如何让几个变量共享同一存储区？**
- **联合变量**是占用同一块内存空间的各成员变量的集合。

- 与结构类似，联合类型也是一种构造类型。一个联合类型中包含有多个成员，**这些成员共享共同的存储区域**，但这些成员并不同时存在，联合**存储区域的大小由各个成员中所占字节数最大的成员决定**，在任何时刻，各个成员中**只能有一个成员拥有该存储**。
- 除了用**关键字union**取代**struct**之外，联合类型的定义、联合变量的声明、以及联合成员的引用在语法上**与结构完全相同**。

- 联合变量的声明、初始化及联合成员的引用与结构完全相同，但是**只能对联合的第1个成员进行初始化。**

```
union chl {  
    char c;  
    short h;  
    long l;  
} v = {'9'} ;  
union chl w = {'a'} ;
```

注意：

程序员必须确保访问的联合成员的类型的一致性。即，数据是以何种类型写入的，读取时就要以那种类型来读。

联合的指针

可以声明联合类型的指针。如：

```
union chl v,*pv=&v;
```

- 说明了一个union chl类型的指针pv，并且取出v地址对pv进行初始化，使联合指针pv指向了联合变量v。
- 值得注意的是，联合所有成员的地址和联合变量的地址都是相同的。因为所有成员都是从同一存储空间的边界（低地址）开始存放。
- 但是，不同成员指针值（地址值）的类型是不相同的。

&u	&u.c	&u.h	&u.l	值相同
union chl *	char *	short *	long *	

字段结构与联合的应用



如何访问**16位字**中的高低字节和各二进制位？

- ◆ 定义**8位宽**的字段**byte0**、**byte1**
 - 表示一个**16位字**中的高/低字节
- ◆ 定义**1位宽**的字段**b0~b15**
 - 表示一个**16位字**中的**bit**
- ◆ 定义联合类型
 - 使一个**short变量**、**2个byte字段**与**16个bit字段**共享存储。

例9.12 用字段和联合访问一个16位字中的高低字节和各二进制位。

```
#include<stdio.h>
#define CHAR_BIT 8
struct w16_bytes{
    unsigned short byte0:8; /* byte0: 低字节*/
    unsigned short byte1:8; /* byte1: 高字节 */
};
struct w16_bits{
    unsigned short b0:1,b1:1,b2:1,b3:1,b4:1,b5:1,b6:1,b7:1,
        b8:1,b9:1,b10:1,b11:1,b12:1,b13:1,b14:1,b15:1;
};
```

union w16{ /* 短整型变量、结构成员byte、结构成员bit共享共同的存储 */

short

i;

struct w16_bytes

byte;

struct w16_bits

bit;

};

void main(void)

{

union w16 w={0}; /* i为0; byte0和byte1也为0; b0~b15皆为0 */

void bit_print(short); /* 函数原型 */

w.bit.b9=1; /* 相当于byte1为2 */

w.bit.b10=1; /* 相当于byte1为6 */

w.byte.byte0='b';

printf("w.i=0x%x\n",w.i); /* 按整型解释、输出共享存储的内容 */

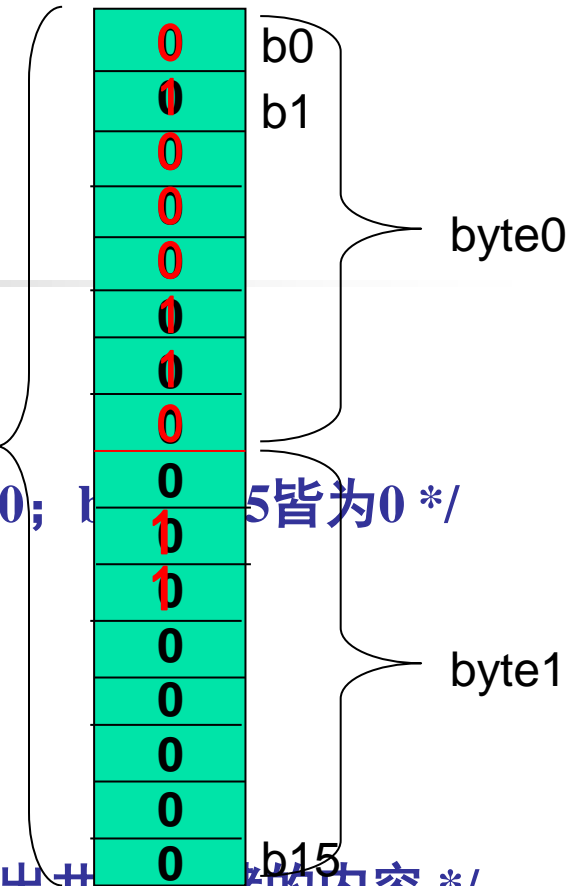
bit_print(w.i); /* 从高到低，逐位输出共享存储的各bit值 */

}

w.i

w.byte

w.bit



运行结果:

w.i=0x662

00000110 01100010