第七十一节： 结构体的内存和赋值。

【71.1　结构体的内存生效。】

上一节讲到结构体有三道标准工序“造模”和“生成”和“调用”，那么，结构体在哪道工序的时候才会开始占用内存(或者说内存生效)？答案是在第二道工序“生成”（或者说定义）的时候才产生内存开销。第一道工序仅“造模”不“生成”是不会产生内存的。什么意思呢？请看下面的例子。

第一种情况：仅“造模”不“生成”。

```
struct StructMould     //“造模”
{
   unsigned char  u8Data_A;
   unsigned char  u8Data_B;
};
```

分析：这种情况是没有内存开销的，尽管你已经写下了数行代码，但是 C 编译器在翻译此代码的时候，它会识别到你偷工减料仅仅“造模”而不“生成”新变量，此时 C 编译器会把你这段代码忽略而过。

第二种情况：先“造模”再“生成”。

```
struct StructMould     //“造模”
{
   unsigned char  u8Data_A;
   unsigned char  u8Data_B;
};

struct StructMould  GtMould_1;  //“生成”一个变量 GtMould_1。占用 2 个字节内存
struct StructMould  GtMould_2;  //“生成”一个变量 GtMould_2。占用 2 个字节内存
```

分析：这种情况才会占用内存。你“生成”变量越多，占用的内存就越大。像本例子，“生成”了两个变量 GtMould_1 和 GtMould_2，一个变量占用 2 个字节，两个就一共占用了 4 个字节。结论：内存的占用是跟变量的“生成”有关。

【71.2　结构体的内存对齐。】

什么是对齐？为了确保内存的地址能整除某个“对齐倍数”（比如 4）。比如以 4 为“对齐倍数”，在地址 0 存放一个变量 a,因为地址 0 能整除“对齐倍数”4，所以符合“地址对齐”，接着往下再存放第二个变量 b，紧接着的地址 1 不能整除“对齐倍数”4，此时，为了内存对齐，本来打算把变量 b 放到地址 1 的，现在就要更改挪到地址 4 才符合“地址对齐”，这就是内存对齐的含义。“对齐倍数”是什么？“对齐倍数”就是单片机的位数除以 8。比如 8 位单片机的“对齐倍数”是 1（8 除以 8）,16 位单片机是 2（16 除以 8），32 位单片机是 4（32 除以 8）。本教材所用的单片机是 8 位的 51 内核单片机，因此“对齐倍数”是 1。1 是可以被任何整数整除的，因此，8 位单片机在结构体的使用上被内存对齐的“干扰”是最小的。

为什么要对齐？单片机内部硬件层面一条指令处理的数据宽度是固定的，比如，因为一个字节是 8 位，所以，8 位的单片机一次处理的数据宽度是 1 个字节（8 除以 8 等于 1），16 位的单片机一次处理的数据宽度是 2 个字节（16 位除以 8 位等于 2），32 位的单片机一次处理的数据宽度是 4 个字节（32 位除以 8 位等于 4），如果字节不对齐，本来单片机一个指令能处理的数据可能就要分解成 2 个指令甚至更多的指令，所以 C 编译器为了让单片机处于最佳状态，在某些情况就会涉及内存对齐，结构体就涉及到内存对齐。

结构体的内存对齐表现在哪里呢？请看下面两个例子：

第一个例子：8 位单片机。

```
struct StructMould_1     //“造模”
{
   unsigned char  u8Data;     //一个 unsigned char 占用 1 个字节。
   unsigned long  u32Data;    //一个 unsigned long 占用 4 个字节。
};

struct StructMould_1  GtMould_1;  //占用多少个字节内存呢？
```

分析：GtMould_1 这个变量占用多少个内存字节呢？假设 GtMould_1 的首地址是 0，那么地址 0 就存放成员 u8Data，u8Data 占用 1 个字节，所以接下来的地址是 1（0+1），问题来了，地址 1 能直接存放占用 4 个字节的成员 u32Data 吗？因为 8 位单片机的“对齐倍数”是 1（8 除以 8），那么地址 1 显然是可以整除“对齐倍数”1 的，因此，地址 1 是可以果断存储 u32Data 成员的。因此，GtMould_1 占用的总字节数是 5（1+4），也就是 u8Data 和 u32Data 两者所占字节数之和。

第二个例子：32 位单片机。

```
struct StructMould_1     //“造模”
{
   unsigned char  u8Data;     //一个 unsigned char 占用 1 个字节。
   unsigned long  u32Data;    //一个 unsigned long 占用 4 个字节。
};

struct StructMould_1  GtMould_1;  //占用多少个字节内存呢？
```

分析：GtMould_1 这个变量占用多少个内存字节呢？假设 GtMould_1 的首地址是 0，那么地址 0 就存放成员 u8Data，u8Data 占用 1 个字节，所以接下来的地址是 1（0+1），那么问题来了，地址 1 能直接存放占用 4 个字节的成员 u32Data 吗？不能。因为 32 位单片机的“对齐倍数”是 4（32 除以 8），那么地址 1 显然是不可以整除“对齐倍数”4 的，因此，就要把地址 1 更改挪到地址 4 这里才符合“地址对齐”，这样，就意味着多插入了 3 个“填充的字节”，因此，GtMould_1 占用的总字节数是 8（1+3+4），也就是“1 个字节 u8Data，3 个填充字节，4 个 u32Data”三者所占字节数之和。那么问题又来了，如果把结构体内部成员 u8Data 和 u32Data 的位置顺序更改一下，内存容量会有所改变吗？位置顺序更改后如下。

```
struct StructMould_1     //“造模”
```

```
{
   unsigned long  u32Data;    //一个 unsigned long 占用 4 个字节。
   unsigned char  u8Data;     //一个 unsigned char 占用 1 个字节。
};

struct StructMould_1  GtMould_1;  //占用多少个字节内存呢？
```

分析：更改 u8Data 和 u32Data 的位置顺序后，u32Data 在前 u8Data 在后，GtMould_1 这个变量占用多少个内存字节呢？假设 GtMould_1 的首地址是 0，那么地址 0 就存放成员 u32Data，u32Data 占用 4 个字节，所以接下来的地址是 4（0+4），那么问题来了，地址 4 能直接存放占用 1 个字节的成员 u8Data 吗？能。因为 32 位单片机的“对齐倍数”是 4（32 除以 8），那么地址 4 显然是可以整除“对齐倍数”4 的，因此，地址 4 是可以果断存储 u8Data 的。那么，是不是 GtMould_1 就占用 5 个字节呢？不是。因为结构体的内存对齐，还包括另外一条规定，那就是“一个结构体变量所占的内存总容量必须能整除该单片机的“对齐倍数”（单片机的位数除以 8），如果不能，C 编译器就会擅自在最后一个成员的后面插入若干个“填充字节”来满足这个规则”，根据这条规定，计算所得的总容量 5 是不能整除“对齐倍数”4 的，必须再额外填充 3 个字节补足到 8，才能整除“对齐倍数”4，因此，更改顺序后，GtMould_1 还是占用 8 个字节（4+1+3），前 4 个字节是 u32Data，中间 1 个字节是 u8Data，后 3 个字节是“填充字节”。

因为本教程采用的是 8 位的 51 内核单片机，因此，在上述这个例子中，GtMould_1 所占的字节数是符合“第一个例子”的情况，也就是占用 5 个字节。内存对齐是遵守几条严格的规则的，我只列出其中最关键的两条给大家大致阅读一下，有一个印象即可，不强求死记硬背，只需知道“结构体因为存在内存对齐，所以实际内存容量是有可能大于内部各成员类型字节数相加之和，尤其是 16 位或者 32 位这类单片机”就可以了。

第（1）条：结构体内部某个成员相对结构体首地址的偏移地址必须能整除该单片机的“对齐倍数”（单片机的位数除以 8），如果不能，C 编译器就会擅自在各成员之间插入若干个“填充字节”来满足这个规则。

第（2）条：一个结构体变量所占的内存总容量必须能整除该单片机的“对齐倍数”（单片机的位数除以 8），如果不能，C 编译器就会擅自在最后一个成员的后面插入若干个“填充字节”来满足这个规则。

【71.3　如何获取某个结构体变量的内存容量？】

结构体存在内存对齐的问题，就说明它的内存占用情况不会像普通数组那样一目了然，那么，我们编写程序的时候怎么知道某个结构体变量占用了多少个字节数？答案是：用 sizeof 宏函数。比如：

```
struct StructMould_1
{
   unsigned long  u32Data;
   unsigned char  u8Data;
};

struct StructMould_1  GtMould_1;

unsigned long a; //此变量用来获取结构体变量 GtMould_1 所占用的字节总数
void main() //主函数
{
```

```
    a=sizeof(GtMould_1);  //利用宏函数 sizeof 获取结构体变量所占用的字节总数
}
```

【71.4　结构体之间的赋值。】

结构体之间的赋值有两种，第一种是成员之间“一对一”的赋值，第二种是整个结构体之间“面对面”的整体赋值。第一种成员赋值像普通变量赋值那样，没有那么多套路和忌讳，数据传递安全可靠。第二种整个结构体之间赋值在编程体验上带有“一键操作”的快感，但是要注意避开一些“雷区”，首先，整体赋值的前提是必须保证两个结构体变量都是同一个“结构体模板”造出来的变量，不同“模板”的结构体变量之间禁止“整体赋值”，其次，哪怕是“同一个模板”的结构体变量，也并不是所有的“同模板结构体”变量都能实现整个结构体之间的直接赋值，只有在结构体内部成员比较简单的情况下才适合“整体赋值”，如果结构体内部包含有“指针”或者“字符串”或者“其它结构体中的结构体”，这类情况就比较复杂，这时建议大家绕开有“雷区”的“整体赋值”而直接选用安全可靠的“成员赋值”。什么是“成员赋值”什么是“整体赋值”？请看下面两个例子。

第一种：成员赋值。把结构体变量 GtMould_2_A 赋值给 GtMould_2_B。

```
struct StructMould_2     //“造模”
{
   unsigned long  u32Data;
   unsigned char  u8Data;
};

struct StructMould_2  GtMould_2_A;  //生成第 1 个结构体变量
struct StructMould_2  GtMould_2_B   //生成第 2 个结构体变量

void main() //主函数
{
     //先给 GtMould_2_A 赋初值。
     GtMould_2_A.u32Data=1;
     GtMould_2_A.u8Data=2;

     //通过“成员赋值”，把结构体变量 GtMould_2_A 赋值给 GtMould_2_B。
     GtMould_2_B.u32Data=GtMould_2_A.u32Data;    //成员之间“一对一”的赋值
     GtMould_2_B.u8Data=GtMould_2_A.u8Data;      //成员之间“一对一”的赋值
}
```

第二种：整体赋值。把结构体变量 GtMould_2_A 赋值给 GtMould_2_B。

```
struct StructMould_2     //“造模”
{
   unsigned long  u32Data;
   unsigned char  u8Data;
};
```

```
struct StructMould_2  GtMould_2_A;  //生成第 1 个结构体变量
struct StructMould_2  GtMould_2_B   //生成第 2 个结构体变量

void main() //主函数
{
     //先给 GtMould_2_A 赋初值。
     GtMould_2_A.u32Data=1;
     GtMould_2_A.u8Data=2;

     //通过“整体赋值”，把结构体变量 GtMould_2_A 赋值给 GtMould_2_B。
     GtMould_2_B=GtMould_2_A;   //整体之间“一次性”的赋值
}
```

上述例子中的整体赋值，是因为结构体内部的数据比较“简单”，没有包含“指针”或者“字符串”或者“其它结构体中的结构体”这类数据成员，如果包含这类成员，建议大家不要用整体赋值。比如遇到以下这类结构体就建议大家直接用安全可靠的“成员赋值”：

```
struct StructMould     //“造模”
{
   unsigned char u8String[]=”String”;  //字符串
   unsigned char  *pu8Data;   //指针
   struct StructOtherMould GtOtherMould;  //结构体中的结构体
};
```

【71.5　例程练习和分析。】

现在编写一个练习的程序：

```
/*---C 语言学习区域的开始。-----------------------------------------------*/

struct StructMould_1     //“造模”
{
   unsigned long  u32Data;    //一个 unsigned long 占用 4 个字节。
   unsigned char  u8Data;     //一个 unsigned char 占用 1 个字节。
};

struct StructMould_2     //“造模”
{
   unsigned char  u8Data;
   unsigned long  u32Data;
};

struct StructMould_1  GtMould_1;  //占用多少个字节内存呢？
```

```
    struct StructMould_2  GtMould_2_A;
    struct StructMould_2  GtMould_2_B;

    unsigned long a; //此变量用来获取结构体变量 GtMould_1 所占用的字节总数

void main() //主函数
{
     a=sizeof(GtMould_1);  //利用宏函数 sizeof 获取结构体变量 GtMould_1 所占用的字节总数

     //先给 GtMould_2_A 赋初值。
     GtMould_2_A.u32Data=1;
     GtMould_2_A.u8Data=2;

     //通过“整体赋值”，把结构体变量 GtMould_2_A 赋值给 GtMould_2_B。
     GtMould_2_B=GtMould_2_A;   //整体之间“一次性”的赋值

     View(a);    //把 a 发送到电脑端观察
     View(GtMould_2_B.u32Data);   //把结构体成员 GtMould_2_B.u32Data 发送到电脑端观察
     View(GtMould_2_B.u8Data);   //把结构体成员 GtMould_2_B.u8Data 发送到电脑端观察

    while(1)
    {
    }
}
/*---C 语言学习区域的结束。----------------------------------------------*/
```

在电脑串口助手软件上观察到的程序执行现象如下：

```
开始...

第 1 个数
十进制:5
十六进制:5
二进制:101

第 2 个数
十进制:1
十六进制:1
二进制:1

第 3 个数
十进制:2
十六进制:2
```

二进制:10

分析：

GtMould_1 所占的字节数 a 为 5。

GtMould_2_B 的结构体成员 GtMould_2_B.u32Data 为 1。

GtMould_2_B 的结构体成员 GtMould_2_B.u8Data 为 2。

【71.6　如何在单片机上练习本章节 C 语言程序？】

直接复制前面章节中第十一节的模板程序，练习代码时只需要更改“C 语言学习区域”的代码就可以了，其它部分的代码不要动。编译后，把程序下载进带串口的 51 学习板，通过电脑端的串口助手软件就可以观察到不同的变量数值，详细方法请看第十一节内容。