第七十四节： 结构体指针在函数接口处的频繁应用。

【74.1　重温“函数的接口参数”。】

函数的接口参数主要起到标识的作用。比如：

一个加法函数：

```
unsigned char add(unsinged char a,unsigned char b)
{
    return (a+b);
}
```

这里的 a 和 b 就是接口参数，它的作用是告诉人们，你把两个加数分别代入 a 和 b，返回的就是你要的加法运算结果。这里的接口参数就起到入口标识的作用。注意，这句话的关键词是“标识”而不是“入口”，因为函数的“入口”不是唯一的，而是无数条路径。为什么这么说？我们把上面的例子改一下，改成全局变量，例子如下:

一个加法函数：

```
unsinged char a;  //加数
unsigned char b; //加数
unsigned char c;  //和
void add(void)
{
    c=a+b;
}
```

上述例子中，尽管我用“两个”void（空的）关键词把原来加法函数的入口（接口参数）和出口（return 返回）都堵得死死的，但是，全局变量是无法阻挡的，它进入一个函数的内部不受任何限制，也就是说，我们做项目的时候，如果把所有函数的接口参数和返回都改成 void 类型，所有的信息传递都改用全局变量，这样也是可以勉强把项目做完成的。但是，如果真的把所有函数的接口参数都改成 void，全部靠全局变量来传递信息，那么最大的问题是函数多了之后，阅读非常不方面，你每看到一个被调用的函数，你不能马上猜出它大概跟哪些全局变量发生了关联，你必须一个一个的去查该函数的源代码才能理清楚，针对这个问题，C 语言的设计者，给了函数非常丰富的接口参数，最理想的函数是：你把凡是与此函数相关的全局变量都经过接口参数的入口才进入到函数内部，尽量把接口参数的入口看作是函数的唯一合法入口（尽管不是唯一也不是必须），这样只要看函数的接口参数就知道这个函数跟哪些全局变量有关，函数的输入输出就非常清晰明了。但是问题又来了，如果有多少个全局变量就开多少个接口参数，接口参数就会变得非常多，接口参数多了，函数的门面就非常难看，无异于把本来应该“小而窄”的接口设在“宽而广”的平原上，还不如直接用原来那种全局变量强行进入呢。那么，要解决这个问题怎么办？本节的主角“结构体指针”可以解决这个问题。

【74.2　结构体指针在函数接口处的频繁应用。】

当函数的接口参数非常多的时候，可以把 N 个相关的全局变量“打包”成一个结构体数据，碰到函数接口的时候，可以通过“结构体指针”以“包”为单位的方式进入，这样就可以让函数的接口参数看起来非常少，这种方法，是很多 32 位单片机的库函数一直在用的方法，它最重要的好处是简化入口的通道数量。你想想，32 位单片机有那么多寄存器，如果没有这种以“结构体指针”为接口参数的方式，它的入口可能需要

几十个接口参数，那岂不是非常麻烦？库函数设计的成败与否，本来就在于接口的设计合不合理，“结构体指针作为函数接口参数”在此场合就显得特别有价值，使用了这种方法，函数与全局变量之间，它们的关联脉络再也不用隐藏起来，并且可以很清晰的表达清楚。现在举一个例子，比如有一个函数，要实现把 5 个全局变量“自加 1”的功能，分别使用两种接口参数来实现，例子如下：

第一种方式：有多少个全局变量就开多少个接口参数。

```
    //函数的声明
    void Add_One( unsigned char *pu8Data_1,  //第 1 个接口参数
                  unsigned char *pu8Data_2,  //第 2 个接口参数
                  unsigned char *pu8Data_3,  //第 3 个接口参数
                  unsigned char *pu8Data_4,  //第 4 个接口参数
                  unsigned char *pu8Data_5);  //第 5 个接口参数

    //5 个全局变量的定义
    unsigned char a;
    unsigned char b;
    unsigned char c;
    unsigned char d;
    unsigned char e;

    //函数的定义
    void Add_One( unsigned char *pu8Data_1,  //第 1 个接口参数
                  unsigned char *pu8Data_2,  //第 2 个接口参数
                  unsigned char *pu8Data_3,  //第 3 个接口参数
                  unsigned char *pu8Data_4,  //第 4 个接口参数
                  unsigned char *pu8Data_5)  //第 5 个接口参数
    {
        *pu8Data_1=(*pu8Data_1)+1;   //实现自加 1 的功能
        *pu8Data_2=(*pu8Data_2)+1;
        *pu8Data_3=(*pu8Data_3)+1;
        *pu8Data_4=(*pu8Data_4)+1;
        *pu8Data_5=(*pu8Data_5)+1;
    }

void main()
{
    //5 个全局变量都初始化为 0
    a=0;
    b=0;
```

```
    c=0;
    d=0;
    e=0;

    //函数的调用，实现 5 个变量都“自加 1”的功能。加“&”表示“传址”的方式进入函数内部。
    Add_One(&a,  //第 1 个接口参数
            &b,  //第 2 个接口参数
            &c,  //第 3 个接口参数
            &d,  //第 4 个接口参数
            &e); //第 5 个接口参数
}
```

第二种方式：把 N 个全局变量打包成一个结构体，以“结构体指针”的方式进入函数内部。

```
    //函数的声明
    void Add_One(struct StructMould *ptMould);  //只有 1 个结构体指针，大大减小了接口参数。

    //结构体的“造模”
    struct StructMould
    {
        unsigned char a;
        unsigned char b;
        unsigned char c;
        unsigned char d;
        unsigned char e;
    };

    struct StructMould GtMould;  //生成一个结构体变量，内部包含了 5 个全局变量 a,b,c,d,e。

    //函数的定义
    void Add_One(struct StructMould *ptMould)  //只有 1 个结构体指针，大大减小了接口参数。
    {
        ptMould->a=ptMould->a+1;  //实现“自加 1”的功能。
        ptMould->b=ptMould->b+1;
        ptMould->c=ptMould->c+1;
        ptMould->d=ptMould->d+1;
        ptMould->e=ptMould->e+1;
    }

    void main()
    {
        //5 个全局变量的结构体成员都初始化为 0
        GtMould.a=0;
        GtMould.b=0;
```

```
        GtMould.c=0;
        GtMould.d=0;
        GtMould.e=0;

        //函数的调用，实现 5 个变量都“自加 1”的功能。加“&”表示“传址”的方式进入函数内部。
        Add_One(&GtMould);  //只有 1 个结构体指针，大大减小了接口参数。
}
```

【74.3　例程练习和分析。】

现在编写一个“以结构体指针为函数接口参数”的练习程序。

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

 //函数的声明
    void Add_One(struct StructMould *ptMould);  //只有 1 个结构体指针，大大减小了接口参数。

    //结构体的“造模”
    struct StructMould
    {
        unsigned char a;
        unsigned char b;
        unsigned char c;
        unsigned char d;
        unsigned char e;
    };

    struct StructMould GtMould;  //生成一个结构体变量，内部包含了 5 个全局变量 a,b,c,d,e。

    //函数的定义
    void Add_One(struct StructMould *ptMould)  //只有 1 个结构体指针，大大减小了接口参数。
    {
        ptMould->a=ptMould->a+1;  //实现“自加 1”的功能。
        ptMould->b=ptMould->b+1;
        ptMould->c=ptMould->c+1;
        ptMould->d=ptMould->d+1;
        ptMould->e=ptMould->e+1;
    }

    void main() //主函数
    {
        //5 个全局变量的结构体成员都初始化为 0
        GtMould.a=0;
        GtMould.b=0;
```

```
        GtMould.c=0;
        GtMould.d=0;
        GtMould.e=0;

        //函数的调用，实现 5 个变量都“自加 1”的功能。加“&”表示“传址”的方式进入函数内部。
        Add_One(&GtMould);  //只有 1 个结构体指针，大大减小了接口参数。

       View(GtMould.a);  //在电脑端观察结构体成员 GtMould.a 的数值。
       View(GtMould.b);  //在电脑端观察结构体成员 GtMould.b 的数值。
       View(GtMould.c);  //在电脑端观察结构体成员 GtMould.c 的数值。
       View(GtMould.d);  //在电脑端观察结构体成员 GtMould.d 的数值。
       View(GtMould.e);  //在电脑端观察结构体成员 GtMould.e 的数值。

       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:1
十六进制:1
二进制:1

第 2 个数
十进制:1
十六进制:1
二进制:1

第 3 个数
十进制:1
十六进制:1
二进制:1

第 4 个数
十进制:1
十六进制:1
二进制:1

第 5 个数
```

```
十进制:1
十六进制:1
二进制:1
```

分析：

结构体成员 GtMould.a 的数值是 1。
结构体成员 GtMould.b 的数值是 1。
结构体成员 GtMould.c 的数值是 1。
结构体成员 GtMould.d 的数值是 1。
结构体成员 GtMould.e 的数值是 1。

【74.4　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。