第六十一节： 指针的中转站作用，地址自加法，地址偏移法。

【61.1 指针与批量数组的关系。】

指针和批量数据的关系，更像领导和团队的关系，领导是团队的代表，所以当需要描述某个团队的时候，为了表述方便，可以把由 N 个人组成的团队简化成该团队的一个领导，用一个领导来代表整个团队，此时，领导就是团队，团队就是领导。指针也一样，指针一旦跟某堆数据“绑定”了，那么指针就是这堆数据，这堆数据就是该指针，所以在很多 PC 上位机的项目中，往往也把指针称呼为“句柄”，字面上理解，就是一句话由 N 个文字组成，而“句柄”就是这句话的代表，实际上“句柄”往往是某一堆资源的代表。不管是把指针比喻成“领导”、“代表”还是“句柄”，指针在这里都有“中间站”这一层含义。

【61.2 指针在批量数据的“中转站”作用。】

指针在批量数据处理中，主要是能节省代码容量，而且是非常直观的节省代码容量。为什么能节省代码容量？是因为可以把某些重复性的具体实现的功能封装成指针来操作，请看下面的例子：

程序要求：根据一个选择变量 Gu8Sec 的值，要从三堆数据中选择对应的一堆数据放到数组 Gu8Buffer 里。当 Gu8Sec 等于 1 的时候选择第 1 堆，等于 2 的时候选择第 2 堆，等于 3 的时候选择第 3 堆。也就是“三选一”。

第 1 种实现的方法：没有用指针，最原始的处理方式。如下：

```
code unsigned char Cu8Memory_1[3]={1,2,3};  //第 1 堆数据
code unsigned char Cu8Memory_2[3]={4,5,6};  //第 2 堆数据
code unsigned char Cu8Memory_3[3]={7,8,9};  //第 3 堆数据

unsigned char Gu8Sec=2;  //选择的变量
unsigned char Gu8Buffer[3];  //根据变量来存放对应的某堆数据的数组
unsigned char i; //for 循环用到的变量 i

switch(Gu8Sec)  //根据此选择变量来切换到对应的操作上
{
    case 1:  //第 1 堆
          for(i=0;i<3;i++)   //第 1 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
          {
             Gu8Buffer[i]=Cu8Memory_1[i];
          }
          break;

    case 2:  //第 2 堆
          for(i=0;i<3;i++)  //第 2 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
          {
             Gu8Buffer[i]=Cu8Memory_2[i];
          }
```

```
            break;

    case 3:  //第 3 堆
            for(i=0;i<3;i++)    //第 3 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
            {
               Gu8Buffer[i]=Cu8Memory_3[i];
            }
            break;

}
```

分析：上述程序中，没有用到指针，出现了 3 次 for 循环的“赋值”的“搬运数据”的动作。

第 2 种实现的方法：用指针作为“中间站”。如下：

```
code unsigned char Cu8Memory_1[3]={1,2,3};  //第 1 堆数据
code unsigned char Cu8Memory_2[3]={4,5,6};  //第 2 堆数据
code unsigned char Cu8Memory_3[3]={7,8,9};  //第 3 堆数据

unsigned char Gu8Sec=2;  //选择的变量
unsigned char Gu8Buffer[3];  //根据变量来存放对应的某堆数据的数组
unsigned char i; //for 循环用到的变量 i
const unsigned char *pCu8; //引入一个指针作为“中间站”

switch(Gu8Sec)  //根据此选择变量来切换到对应的操作上
{
    case 1:  //第 1 堆
            pCu8=&Cu8Memory_1[0];  //跟第 1 堆数据“绑定”起来。
            break;

    case 2:  //第 2 堆
            pCu8=&Cu8Memory_2[0];  //跟第 2 堆数据“绑定”起来。
            break;

    case 3:  //第 3 堆
            pCu8=&Cu8Memory_3[0];  //跟第 3 堆数据“绑定”起来。
            break;
}

 for(i=0;i<3;i++)  //第 1 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
 {
     Gu8Buffer[i]=*pCu8;  //把“指针所存的地址的数据”赋值给数组
     pCu8++;  //“指针所存的地址”自加 1，为下一个数据的“赋值”的“搬运”作准备。
 }
```

分析：上述程序中，用到了指针作为中间站，只出现了 1 次 for 循环的“赋值”的“搬运数据”的动作。对比之前第 1 种方法，在本例子中，用了指针之后，程序代码看起来更加高效简洁清爽省容量。在实际项目中，数据量越大的时候，指针这种“优越性”就越明显。

【61.3　指针在书写上另外两种常用写法。】

刚才 61.2 处第 2 个例子中，有一段代码如下：

```
for(i=0;i<3;i++)  //第 1 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
{
    Gu8Buffer[i]=*pCu8;  //把“指针所存的地址的数据”赋值给数组
    pCu8++;  //“指针所存的地址”自加 1，为下一个数据的“赋值”的“搬运”作准备。
}
```

很多高手，喜欢把上面 for 循环内部的那两行代码简化成一行代码，如下：

```
for(i=0;i<3;i++)  //第 1 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
{
    Gu8Buffer[i]=*pCu8++;  //先把“数据”赋值给数组，然后“指针所存的地址”再自加 1。
}
```

上面这种写法也是合法的，而且在高手的代码中常见，据说也是最高效的写法。还有一种是利用“指针的偏移地址”的写法，我常用这种写法，因为感觉这种写法比较直观，而且跟数组的书写很像。如下：

```
for(i=0;i<3;i++)  //第 1 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
{
    Gu8Buffer[i]=pCu8[i];  //这类是“偏移地址”的写法，i 在这里相当于指针的偏移地址。
}
```

这种写法也是跟前面那两种写法在程序实现的功能上是一样的，是等效的，我常用这种写法。

【61.4　指针的“地址自加法”和“地址偏移法”的差别。】

刚才 61.3 处讲了 3 个例子，其中前面的两个例子都是属于“地址自加法”，而最后的那一个是属于“地址偏移法”。它们的根本差别是：“地址自加法”的时候，“指针所存的地址”是变动的；而“地址偏移法”的时候，“指针所存的地址”是不变的，“指针所存的地址”的“不变”的属性，就像某个原点，原点再加上偏移，就可以寻址到某个新的 RAM 地址所存的数据。例子如下：

第 1 种：“地址自加法”：

```
pCu8=&Cu8Memory_2[0];  //假设赋值后，此时“指针所存的地址”是 RAM 的地址 4。
for(i=0;i<3;i++)
{
```

```
        Gu8Buffer[i]=*pCu8++;  //先把“数据”赋值给数组，然后“指针所存的地址”再自加 1。
    }
```

分析：上述代码，等程序执行完 for 循环后，指针所存的地址还是 RAM 地址 4 吗？不是。因为它是变动的，经过 for 循环，“指针所存的地址”自加 3 次后，此时“所存的 RAM 地址”从原来的 4 变成了 7。

第 2 种：“地址偏移法”：

```
    pCu8=&Cu8Memory_2[0];  //假设赋值后，此时“指针所存的地址”是 RAM 的地址 4。
    for(i=0;i<3;i++)
    {
       Gu8Buffer[i]=pCu8[i];  //这类是“偏移地址”的写法，i 在这里相当于指针的偏移地址。
    }
```

分析：上述代码，等程序执行完 for 循环后，指针所存的地址还是 RAM 地址 4 吗？是的。因为它存的地址是不变的，变的只是偏移地址 i。此时“指针所存的地址”就像“原点”一样具有“绝对地址”的“参考点”的属性。

【61.5　例程练习和分析。】

现在编一个练习程序。

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

    code unsigned char Cu8Memory_1[3]={1,2,3};  //第 1 堆数据
    code unsigned char Cu8Memory_2[3]={4,5,6};  //第 2 堆数据
    code unsigned char Cu8Memory_3[3]={7,8,9};  //第 3 堆数据

    unsigned char Gu8Sec=2;  //选择的变量
    unsigned char Gu8Buffer[3];  //根据变量来存放对应的某堆数据的数组
    unsigned char i; //for 循环用到的变量 i
    const unsigned char *pCu8; //引入一个指针作为“中间站”

void main() //主函数
{

    switch(Gu8Sec)  //根据此选择变量来切换到对应的操作上
    {
        case 1:  //第 1 堆
              pCu8=&Cu8Memory_1[0];  //跟第 1 堆数据“绑定”起来。
              break;

        case 2:  //第 2 堆
              pCu8=&Cu8Memory_2[0];  //跟第 2 堆数据“绑定”起来。
```

```
            break;

        case 3:  //第 3 堆
            pCu8=&Cu8Memory_3[0];  //跟第 3 堆数据“绑定”起来。
            break;
    }

    // for(i=0;i<3;i++)  //第 1 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
    // {
    //     Gu8Buffer[i]=*pCu8++; //先把“数据”赋值给数组，然后“指针所存的地址”再自加 1。
    // }

    for(i=0;i<3;i++)  //第 1 次出现 for 循环，用来实现“赋值”的“搬运数据”的动作。
    {
        Gu8Buffer[i]=pCu8[i];  //这类是“偏移地址”的写法，i 在这里相当于指针的偏移地址。
    }

    View(Gu8Buffer[0]);  //把第 1 个数 Gu8Buffer[0]发送到电脑端的串口助手软件上观察。
    View(Gu8Buffer[1]);  //把第 2 个数 Gu8Buffer[1]发送到电脑端的串口助手软件上观察。
    View(Gu8Buffer[2]);  //把第 3 个数 Gu8Buffer[2]发送到电脑端的串口助手软件上观察。

    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:4
十六进制:4
二进制:100

第 2 个数
十进制:5
十六进制:5
二进制:101

第 3 个数
十进制:6
十六进制:6
```

二进制:110

分析：

Gu8Buffer[0]为 4。

Gu8Buffer[1]为 5。

Gu8Buffer[2]为 6。

【61.6　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。