第六十节： 指针在变量(或常量)中的基础知识。

【60.1 指针与普通变量的对比。】

普通变量和指针都是变量，都要占用 RAM 资源。普通变量的 unsigned char 类型占用 1 个字节，unsigned int 类型占用 2 个字节，unsigned long 类型占用 4 个字节。但是指针不一样，指针是一种特殊的变量，unsigned char*, unsigned int*, unsigned long*这三类指针都是一样占用 4 个字节。指针是普通变量的载体，平时我们处理普通变量，都是可以“直接”操作普通变量本身。而学了指针之后，我们就多一种选择，可以通过指针这个载体来“间接”操作某个普通变量。“直接”不是比“间接”更好更高效吗？为什么要用“间接”？其实在某些场合，指针的“间接”操作更加灵活更加高效，这个要看具体的应用。

指针既然是普通变量的“载体”，那么普通变量就是“物”。“载体”与“物”之间可以存在一对多的关系。也就是说，一个篮子（载体），可以盛放鸡蛋（物），也可以盛放青菜（物），也可以盛放水果（物）。但是，在这里，一个篮子在一个时间段内，只能承载一种物品，如果想承载其它物品，必须先把当前物品“卸”下来，然后再“装”其它物品”。这里有两个关键动作“装”和“卸”，就是指针在处理普通变量时的“绑定”，某个指针与某个变量发生“绑定”，就已经包含了先“卸”后“装”这两个动作在其中。

题外话多说一句，刚才提到，unsigned int 类型占用 2 个字节，这个是在 C51 编译器下的情况。如果是在 stm32 单片机的编译器下，unsigned int 类型是占用 4 个字节。而“凡是指针都是 4 个字节”，这个描述仅仅适用于 32 位以下的单片机编译器（包括 8 位的单片机），而在某些 64 位的 PC 机，指针可能是 8 个字节，这些内容大家只要有个大概的了解即可。

【60.2 指针的定义。】

跟普通变量一样，指针也必须先定义再使用。为了与普通变量区分开来，指针在定义的时候多加了一个星号“*”，例子如下：

```
unsigned char* pu8;   //针对 unsigned char 类型变量的指针。凡是指针都是占 4 个字节！
unsigned int* pu16;   //针对 unsigned int 类型变量的指针。凡是指针都是占 4 个字节！
unsigned long* pu32;  //针对 unsigned long 类型变量的指针。凡是指针都是占 4 个字节！
```

既然指针都是 4 个字节，为什么还要区分 unsigned char*，unsigned int* pu16，unsigned long* pu32 这三种类型？因为指针是为普通变量（或常量）而生，所以要根据普通变量（或常量）的类型定义对应的指针。

【60.3 指针与普通变量是如何关联和操作的？】

指针在操作某个变量的时候，必须先跟某个变量关联起来，这里的关联就是“绑定”。“绑定”后，才可以通过指针这个“载体”来“间接”操作变量。指针与普通变量在“绑定”的时候，需要用到“&”这个符号。例子如下：

```
unsigned char* pu8;   //针对 unsigned char 类型变量的指针。凡是指针都是占 4 个字节！
unsigned char a=0;    //普通的变量。
pu8=&a;  //指针与普通变量发生关联（或者说绑定）。
*pu8=2;  //通过指针这个载体来处理 a 这个变量，此时 a 从原来的 0 变成了 2。
```

【60.4　指针处理“批量数据”的基础知识。】

之所以有通过载体来“间接”操作普通变量的存在价值，其中很重要的原因是指针在处理“批量数据”时特别给力，这里的“批量数据”是有条件的，要求这些数据的地址必须挨家挨户连起来的，不能是零零散散的“散户”，比如说，数组就是由一堆在 RAM 空间里地址连续的变量组合而成，指针在很多时候就是为数组而生的。先看一个例子如下：

```
unsigned char* pu8;   //针对 unsigned char 类型变量的指针。凡是指针都是占 4 个字节！
unsigned char Buffer[3];    //普通的数组，内含 3 个变量，它们地址是相连的。

pu8=&Buffer[0];  //指针与普通变量 Buffer[0]发生关联（或者说绑定）。
*pu8=1;          //通过指针这个载体来处理 Buffer[0]这个变量，此时 Buffer[0]变成了 1。

pu8=&Buffer[1];  //指针与普通变量 Buffer[1]发生关联（或者说绑定）。
*pu8=2;          //通过指针这个载体来处理 Buffer[1]这个变量，此时 Buffer[1]变成了 2。

pu8=&Buffer[2];  //指针与普通变量 Buffer[2]发生关联（或者说绑定）。
*pu8=3;          //通过指针这个载体来处理 Buffer[2]这个变量，此时 Buffer[2]变成了 3。
```

分析：上述例子中，并没有体现出指针的优越性，因为数组有 3 个元素，居然要绑定了 3 次，如果数组有 1000 个元素，难道要绑定 1000 次？显然这样是繁琐低效不可取的。而要发挥指针的优越性，我们现在必须深入了解一下指针的本质是什么，指针跟普通变量发生“绑定”的本质是什么。普通变量由“地址”和“地址所装的数据”构成，指针是特殊的变量，它是由什么构成呢？其实，指针是由“地址”和“地址所装的变量（或常量）的地址”组成。很明显，一个重要的区别是，普通变量装的数据，而指针装的是地址。正因为指针装的是地址，所以指针可以有两种选择，第一种可以处理“装的地址”，第二种可以处理“装的地址的所在数据”，这两种能力，就是指针的精华和本质所在，也是跟普通变量的区别所在。那么指针处理“装的地址”的语法是什么样子的？请看例子如下：

```
unsigned char* pu8;   //针对 unsigned char 类型变量的指针。凡是指针都是占 4 个字节！
unsigned char Buffer[3];   //普通的数组，内含 3 个变量，它们地址是相连的。

pu8=&Buffer[0];  //处理“装的地址”。把 Buffer[0]变量的地址装在指针这个载体里。
*pu8=1;      //处理“装的地址的所在数据”。此时 Buffer[0]变成了 1。

pu8++;       //处理“装的地址”。这里是“地址”自加 1，相当于指针此时装的是 Buffer[1]的地址。
*pu8=2;      //处理“装的地址的所在数据”。此时 Buffer[1]变成了 2。

pu8++;       //处理“装的地址”。这里是“地址”自加 1，相当于指针此时装的是 Buffer[2]的地址。
*pu8=3;      //处理“装的地址的所在数据”。此时 Buffer[2]变成了 3。
```

上述例子中，利用“地址”自加 1 的操作，省去了 2 条赋值式的“绑定”操作（比如像 pu8=&Buffer[0]这类语句），因此“绑定”本质其实就是更改指针所装的“变量（或常量）的地址”的操作。此例子中虽然还没体现了出指针在数组处理时的优越性，但是利用指针处理“装的地址”这项功能，在实际项目中很容易

发现它的好处。

【60.5　指针与数组关联（绑定）时省略“&和下标[0]”的写法。】

指针与数组关联的时候，通常是跟数组的第 0 个元素的地址关联，此时，可以把数组的“&和下标[0]”省略，比如：

```
unsigned char* pu8;
unsigned char Buffer[3];
pu8=Buffer;     //此行代码省略了“&和下标[0]”，等效于 pu8=&Buffer[0];
```

【60.6　带 const 关键字的常量指针。】

指针也可以跟常量关联起来，处理常量，但是常量只能“读”不能“写”，所以通过指针操作常量的时候也是只能“读”不能“写”。操作常量的指针用 const 关键词修饰，强调此指针只有“读”的操作。例子如下：

```
const unsigned char* pCu8;   //常量指针
code char Cu8Buffer[3]={5,6,7};   //常量数组
unsigned char b;
unsigned char c;
unsigned char d;

pCu8=Cu8Buffer; //此行代码省略了“&和下标[0]”，等效于 pCu8=&Cu8Buffer[0];
b=*pCu8;        //读“装的地址的所在数据”。b 等于 5。

pCu8++;         //所装的地址自加 1，跟 Cu8Buffer[1]关联
c=*pCu8;        //读“装的地址的所在数据”。c 等于 6。

pCu8++;         //所装的地址自加 1，跟 Cu8Buffer[2]关联
d=*pCu8;        //读“装的地址的所在数据”。d 等于 7。
```

【60.7　例程练习和分析。】

现在编一个练习程序来熟悉指针的基础知识。

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

unsigned char* pu8;         //针对 unsigned char 类型变量的指针。凡是指针都是占 4 个字节！
unsigned char a=0;          //普通的变量。
unsigned char Buffer[3];    //普通的数组，内含 3 个变量，它们地址是相连的。

const unsigned char* pCu8;   //常量指针
code char Cu8Buffer[3]={5,6,7};   //常量数组
```

```
    unsigned char b;
    unsigned char c;
    unsigned char d;


void main() //主函数
{

    pu8=&a;  //指针与普通变量发生关联（或者说绑定）。
    *pu8=2;  //通过指针这个载体来处理 a 这个变量，此时 a 从原来的 0 变成了 2。


    pu8=&Buffer[0];  //处理“装的地址”。把 Buffer[0]变量的地址装在指针这个载体里。
    *pu8=1;      //处理“装的地址的所在数据”。此时 Buffer[0]变成了 1。

    pu8++;       //处理“装的地址”。这里是“地址”自加 1，相当于指针此时装的是 Buffer[1]的地址。
    *pu8=2;      //处理“装的地址的所在数据”。此时 Buffer[1]变成了 2。

    pu8++;       //处理“装的地址”。这里是“地址”自加 1，相当于指针此时装的是 Buffer[2]的地址。
    *pu8=3;      //处理“装的地址的所在数据”。此时 Buffer[2]变成了 3。

    pCu8=Cu8Buffer; //此行代码省略了“&和下标[0]”，等效于 pCu8=&Cu8Buffer[0];
    b=*pCu8;         //读“装的地址的所在数据”。b 等于 5。

    pCu8++;          //所装的地址自加 1，跟 Cu8Buffer[1]关联
    c=*pCu8;         //读“装的地址的所在数据”。c 等于 6。

    pCu8++;          //所装的地址自加 1，跟 Cu8Buffer[2]关联
    d=*pCu8;         //读“装的地址的所在数据”。d 等于 7。

    View(a);           //把第 1 个数 a 发送到电脑端的串口助手软件上观察。
    View(b);           //把第 2 个数 b 发送到电脑端的串口助手软件上观察。
    View(c);           //把第 3 个数 c 发送到电脑端的串口助手软件上观察。
    View(d);           //把第 4 个数 d 发送到电脑端的串口助手软件上观察。
    View(Buffer[0]);  //把第 5 个数 Buffer[0]发送到电脑端的串口助手软件上观察。
    View(Buffer[1]);  //把第 6 个数 Buffer[1]发送到电脑端的串口助手软件上观察。
    View(Buffer[2]);  //把第 7 个数 Buffer[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:2
十六进制:2
二进制:10

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

第 4 个数
十进制:7
十六进制:7
二进制:111

第 5 个数
十进制:1
十六进制:1
二进制:1

第 6 个数
十进制:2
十六进制:2
二进制:10

第 7 个数
十进制:3
十六进制:3
二进制:11
```

分析：

a 为 2。

b 为 5。

c 为 6。

d 为 7。
Buffer[0]为 1。
Buffer[1]为 2。
Buffer[2]为 3。

【60.8　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。