

## 第二十二节：连减、自减、自减简写、自减 1。

### 【22.1 连减。】

上一节的减法例子中，右边的减数只有两个。实际上，C 语言规则没有限制减数的个数，它的通用格式如下：

“保存变量” = “减数 1” - “减数 2” - “减数 3” - ... - “减数 N”；

当右边的减数个数超过两个的时候（这里暂时把平时所说的被减数也归类为减数），这种情况就是“连减”。每个减数的属性没有限定，可以是常量，也可以是变量。比如：

```
a=68-3-15;    //减数全部是常量。
b=q-x-y-k;    //减数全部是变量。
c=63-x-5-k;    //减数有的是常量，有的是变量。
```

连减的运行顺序是，赋值符号“=”右边的减数挨个相减，把每一次的运算结果放在一个临时的隐藏中间变量里，这个隐藏的变量我们看不到，是单片机系统内部参与运算时的专用寄存器，等右边所有减数连减的计算结果出来后，再把隐藏变量所保存的计算结果赋值给左边的“保存变量”。

### 【22.2 自减、自减简写、自减 1。】

什么是自减？当赋值符号“=”右边的第 1 个减数是“保存变量”本身时（这里暂时把平时所说的被减数也归类为减数），这种情况就是“自减”。自减在程序里有一个特点，只要第 2 个减数不为 0，那么每执行一次这行代码，“保存变量”本身就会减小一次，不断执行这行代码，“保存变量”本身就会不断减小，而每次的减小量就取决于赋值符号“=”右边从第 2 个减数开始后面所有减数之和。自减的常见格式如下：

“保存变量” = “保存变量” - “减数 2”；  
“保存变量” = “保存变量” - “减数 2” - “减数 3” - ... - “减数 N”；

在这类自减计算式中，当只有右边的第 1 个减数是“保存变量”本身时，那么上述自减计算式可以简写成如下格式：

“保存变量” -= “减数 2”；  
“保存变量” -= （“减数 2” + “减数 3” + ... + “减数 N”）；

这种格式就是“自减简写”。现在举几个例子如下：

```
d-=6;    //相当于 d=d-6;
e-=x;    //相当于 e=e-x;
f-=18-y-k; //相当于 f=f-(18-y-k);
```

这些例子都是很常规的自减简写，再跟大家讲一种很常用的特殊简写。当右边只有两个减数，而第 1 个减数是“保存变量”，第 2 个减数是常数 1 时，格式如下：

“保存变量” = “保存变量” - 1；

这时候，可以把上述格式简写成如下两种格式：

“保存变量” --；  
-- “保存变量”；

这两种格式也是俗称的“自减 1”操作。比如：

```
g--;    //相当于 g=g-1 或者 g-=1;
--h;    //相当于 h=h-1 或者 h-=1;
```

自减 1 符号“--”可以在变量的左边，也可以在变量的右边，它们在这里本质是一样的，没有差别。当然，如果是在循环条件语句中，这时自减 1 符号“--”在左边还是在右边是有一点点微弱的差别，这方面的内容以后再讲。

### 【22.3 例程练习和分析。】

现在我们编写一个程序来验证上面讲到的例子：

程序代码如下：

```
/*---C 语言学习区域的开始。-----*/

void main() //主函数
{
    unsigned char a;    //定义一个变量 a, 并且分配了 1 个字节的 RAM 空间。
    unsigned char b;    //定义一个变量 b, 并且分配了 1 个字节的 RAM 空间。
    unsigned char c;    //定义一个变量 c, 并且分配了 1 个字节的 RAM 空间。
    unsigned char d=65; //定义一个变量 d, 并且分配了 1 个字节的 RAM 空间。初始化默认为 65.
    unsigned char e=38; //定义一个变量 e, 并且分配了 1 个字节的 RAM 空间。初始化默认为 38.
    unsigned char f=29; //定义一个变量 f, 并且分配了 1 个字节的 RAM 空间。初始化默认为 29.
    unsigned char g=5;  //定义一个变量 g, 并且分配了 1 个字节的 RAM 空间。初始化默认为 5.
    unsigned char h=5;  //定义一个变量 h, 并且分配了 1 个字节的 RAM 空间。初始化默认为 5.

    unsigned char q=50; //定义一个变量 q, 并且分配了 1 个字节的 RAM 空间。初始化默认为 50.
    unsigned char x=3;  //定义一个变量 x, 并且分配了 1 个字节的 RAM 空间。初始化默认为 3.
    unsigned char y=6;  //定义一个变量 y, 并且分配了 1 个字节的 RAM 空间。初始化默认为 6.
    unsigned char k=2;  //定义一个变量 k, 并且分配了 1 个字节的 RAM 空间。初始化默认为 2.

    //第 1 个知识点：连减。
    a=68-3-15;          //减数全部是常量。a 的结果为：50。
    b=q-x-y-k;          //减数全部是变量。b 的结果为：39。
    c=63-x-5-k;          //减数有的是常量，有的是变量。c 的结果为：53。

    //第 2 个知识点：自减简写。
    d-=6;                //相当于 d=d-6; d 的结果为：59。
    e-=x;                //相当于 e=e-x; e 的结果为：35。
    f-=18-y-k;           //相当于 f=f-(18-y-k); f 的结果为：19。

    //第 3 个知识点：自减 1。
    g--;                 //相当于 g=g-1 或者 g-=1; g 的结果为：4。
    --h;                 //相当于 h=h-1 或者 h-=1; d 的结果为：4。

    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(e);             //把第 5 个数 e 发送到电脑端的串口助手软件上观察。
    View(f);             //把第 6 个数 f 发送到电脑端的串口助手软件上观察。
}
```

```

View(g);          //把第 7 个数 g 发送到电脑端的串口助手软件上观察。
View(h);          //把第 8 个数 h 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:50

十六进制:32

二进制:110010

第 2 个数

十进制:39

十六进制:27

二进制:100111

第 3 个数

十进制:53

十六进制:35

二进制:110101

第 4 个数

十进制:59

十六进制:3B

二进制:111011

第 5 个数

十进制:35

十六进制:23

二进制:100011

第 6 个数

十进制:19

十六进制:13

二进制:10011

```
第 7 个数  
十进制:4  
十六进制:4  
二进制:100
```

```
第 8 个数  
十进制:4  
十六进制:4  
二进制:100
```

分析:

通过实验结果，发现在单片机上的计算结果和我们的分析是一致的。

#### 【22.4 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。