



专题

翻天覆地，呼风唤雨

实际动手创建第一个程序

文/杜 洋

转过篇来，不知不觉中你就已经变成了单片机编程准高手，前提是在本文的基础上。下面的任务就是让你努力摆脱我和任何人对你的约束。但你要知道这是过程并不是目的，每个人都应该有独自编程经验，这样对解决问题和树立自己的编程风格是很有好处的，单独拿出一节文章来介绍也是有价值的，千万不要误解成每一个程序都应该由

来想想我们需要实现的出来：

1. 按键控制开关灯
 2. 按键控制台灯亮
 3. 台灯亮度无级
 4. 不断电的情况
- 上列功能只是最

第四节 《翻天覆地，呼风唤雨》

转过篇来，不知不觉中你就已经变成了单片机编程准高手，前提是在本文的基础上。下面的任务就是让你努力摆脱我和任何人对你的约束。但你要知道这是过程并不是目的，每个人都应该有独自编程经验，这样对解决问题和树立自己的编程风格是很好处的，单独拿出一节文章来介绍也是有价值的，千万不要误解成每一个程序都应该由自己原创，也许在工业革命之前你可以这样做，可是现在则需要你站在巨人的肩膀上看得更高，踏着前人的脚步走得更远，应用现有优秀程序是不二选择。

踏上思程之路

编程很简单，我现在这样认为，虽然两年前我还认为我根本学不会 C 语言。还记得大学毕业后的一次应聘面试，事前我总是把我最优秀的地方准备秀给人家，绞尽脑汁总结出一句经典台词。面试官问：“你都会一些什么编程语言”。我答：“其实用什么语言编程并不重要，它只是一种工具而已，关键还是要看程序的思路 and 结构，就好像写一部小说是用英文还是中文并不重要，重要的是小说中精彩故事情节”。那次面试虽然失败告终，可我一直坚信这一观点。不论你选择什么编程语言都要先设计思路 and 结构，思路清晰反而没什么问题了，我们开始从空白建立程序思路是件相当过瘾的事情。

事先要准备几张白纸、一支铅笔和平静的心情，要知道我们要制作什么，我们的目标是什么，把重要的内容记在纸上，这是日后整理计划书的好方法。在你感觉茫然而不知所措的时候我已经想好了一个方案。依我看我们制作一个具有亮度可调功能的台灯怎么样？名字就叫“调光台灯”。前面我们已经涉及到渐变亮度的闪烁 LED 程序了，制作应该简单多了，而且很实用呀！宁静的夜晚，在自己制作的可调亮度的台灯下细细品读《单片机魔法学校》简直就是爽呆了。东西是好，在制作的开始我们要先设计出方案才行，要知道我们不是先有程序再谈功能的，程序应该是制作的最后环节。先想想我们的制作需要几个部分：电源部分、灯泡部分、单片机控制部分、按键操作部分。再来想想我们所需要的功能，最好是把几条功能特征列出来：

- 1， 按键控制开关灯。
- 2， 按键控制台灯亮度。
- 3， 台灯亮度无级调整。
- 4， 不断电的情况下，开关台灯时的亮度可记忆。

上列功能只是最基本的功能，当然你可以再加一个蜂鸣器，让它可以具有提示音或是唱起音乐还可以加遥控器、加闹钟或是更具天才想象力的功能，而现在我们只就最基本的功能先制作。基本的功能在白纸上列出之后，下一步我们再细分它们：

- 1， 采用一个按键单独控制开关灯，按一下开灯再按关灯。
- 2， 采用“变亮”和“变暗”两个按键控制台灯的亮度，长按时亮度逐渐改变，放开时定格亮度。
- 3， 采用 256 级实现虚拟无级亮度调整。
- 4， 当前亮度数据在单片机的 RAM 中保存，在单片机没有被复位时保证开关灯时记忆先前亮度。

要知道凡是与人有关的设计一定要考虑人性化、简单化和通俗化。在细分功能时要注意这一点，要服从大众的逻辑、让操作简单方便。如果大家都认为文章是从左往右阅读的，而你非要从右往左写，那你就只能留给外星人看了，创

新是为了更优秀，不应该以牺牲人性化为代价。细分功能确定了，我们就可以写一样东西，你会以为是写程序吧，其实应该是使用说明书，当然在细分功能时已经考虑到程序部分的编写，这种评估需要你有一定的编程经验，至少要知道这个功能是可以单片机系统实现的。我们可以用一个 LED 假装灯泡，因为把 LED 转换成一只真正灯泡的电路并不复杂，将 LED 接到单片机的 P1.0 接口，这个电路我们已经有经验。另外需要 3 个按键，一个电源开关，两个亮度调整，这个电路可以用微动开关接在单片机的 3 个 I/O 接口上，微动开关另一端接地即可。采用我们正在学习的 89S 系列单片机和 5V 电源，这就不多说了。然后就是画电路原理图，制作硬件电路，因为本文主要是介绍编程，电路的部分一带而过，图 1 所示是调光台灯的电路原理图，下面的程序就依此硬件编写。

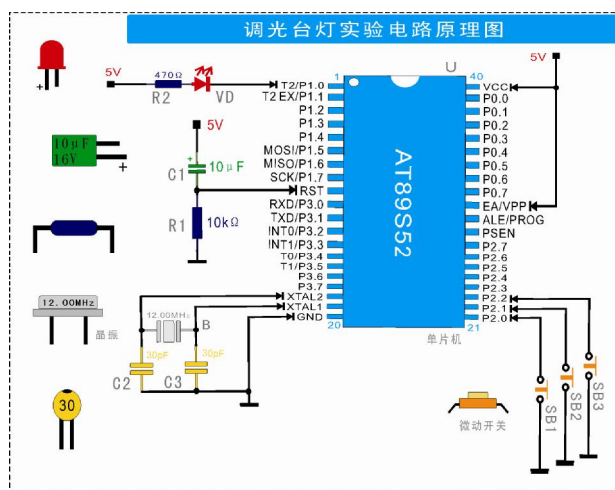


图 1 (调光台灯电路原理图)

现在小说《调光台灯》的精彩故事情节已经确定，只差我们用语言把它写出来。至于用什么语言就看个人爱好了，本文着力 C 语言，并将一直贯彻始终。真的开始写程序的时候就会涉及到另一个问题，就是结构流程。我们要用一个什么样的结构流程来实现精彩故事情节呢，通常情况下程序流程图可以有效的帮助我们，除了大的框架外也有一些细节问题同样需要思考，欲知这些内容敬请继续欣赏。

通达程序流程

我要介绍程序流程图，我的腿开始发抖，脸色也变得苍白，看上去很紧张的样子。是呀，自我学习单片机到现在没有画过几个完整的程序流程图，现在逼上梁山又要介绍，所以有什么言语不周的请多多包涵。要知道不画程序流程图其实是一个很不好的习惯，我是需要自我检讨的，总是认为这没什么用处，最后还是深受其害，我写的程序就有结构混乱、出现过本应该避免的错误。一些计算机专业出身的朋友是不会有这些问题的，只是现在广大单片机爱好者之前都是搞电子技术的，所以程序方面的理论水平自然欠缺，所写的程序也处在基本可靠的境域。我希望单片机初学者可以学习一下计算机基础知识，这可以为你未来的编程之路打好坚实基础。

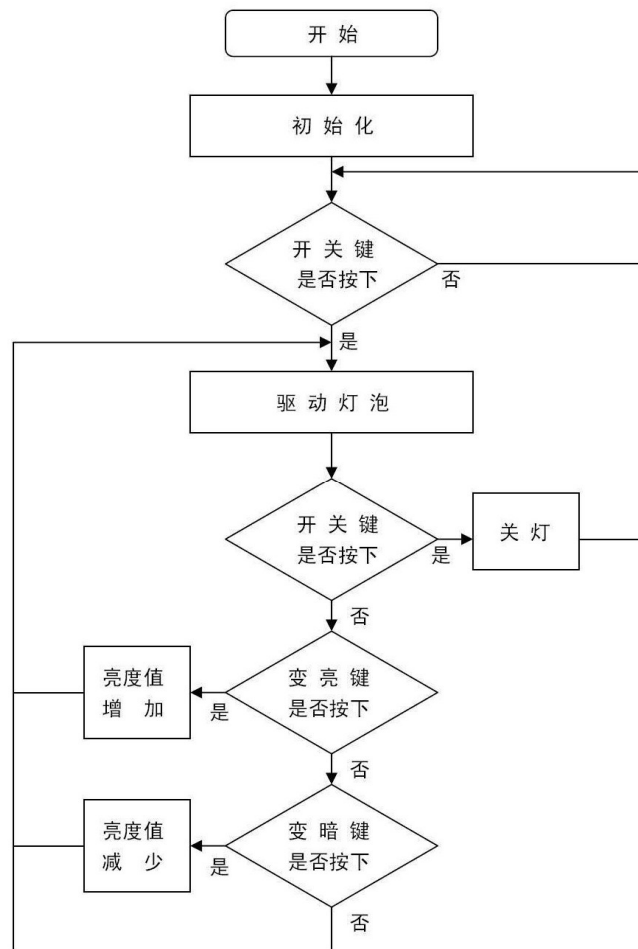


图 2（调光台灯程序流程图）

图 2 所示是调光台灯的程序流程图，其实流程图就是将一个系统的程序分成多个部分，在编程时只要将各部分的程序块写好，再在主循环程序中将各程序块按顺序联系起来。所以说有了流程图之后程序的结构计划也就完成了。绘制程序结构框图也是要靠经验的，当程序写的多了自然就可以总结出流程图的各部分和排列的顺序，而最基本的还是跟着调光台灯的功能思路来写，前文的功能已经确定，程序流程就必须要实现这些功能。通常在接通电源时程序应该是怎样的，之后的什么条件会触发它改变状态，改变状态后的工作又是什么，之后又会有几种状态选择。依据这个思路往下想，在草纸上就会勾勒出靠近完整的图样，检查有多少个循环结构，各部分功能可不可以再细分。最后按照单片机的思路从头到尾把各种可能的状态都走一遍看看还有什么失误或更好的方法。三五回合之后，一个正确、完整的程序流程图诞生了。也许你还没有真正开始写一行程序呢，可以你的编程工作已经完成了一半，而且是最关键的一半，你的程序是否优秀、简洁、实用，到现在已经分出高下。小说《调光台灯》精彩的故事大纲已经确定，余下工作就是用生动的语言书写。

树立编程风格

也许之前你在别的文章中见过 C 语言的程序，是不是有一些地方和本文的程序不同呢，当然我是指风格上的不同。编程编译后的二进制文件（如 HEX）是给单片机看的，只要程序的语法正确，结构正确就在运行时是不会有问题的。就好像我们写信一样，只要让对方知道你写的内容就可以了，至于是宋体字还是楷体字并不重要，只是你的个人习惯。我看过的一些单片机教材上都是在讲程序的内容，从没见到有介绍编程写法的，可能是我看的书不多，不过就

我看过的确实没有。我认为这是漏洞，我现在写稿子都是用电脑打字了，所以大家看不到我手写字的样子，说实话不能说难看，而是相当难看。小学时候，老师教我们写字时就没有注重写字的形体，那时只是要求哪几课的生字词，每个写两行，当时我是贪玩型的，想多一点时间研究小电池和小灯泡，于是急急忙忙的写，最后写的又快又丑。老师照样在作业本上打对勾，因为她和我们都认为只要写完就好，美丑无所谓。大学时给心怡的同学写信，虽然很下功夫可是还是得到人家“字如其人”的客观评价。所以这种观念在我的心中种下，留下难忘的童年阴影。现在的我下意识中对美观非常敏感，我要制作的单片机作品总是会用很多时间来考虑美观问题，包括写单片机程序也是美观优先的。我是希望编程初学者可以切实的注意到这一点，在你的程序可以正常运行也能被别人看懂的同时再注重一下格式和注释信息，这种细致、认真的态度会让你更优秀。

先说一下我的编程目标，看看哪一些观点是你认同的。一，简洁明了，程序文本打开之后首先给人的印象就是干净、利索。二，条理清晰，程序运行的每一步都会一目了然，很容易分清楚、看明白。三，注释意赅，不一定每一条程序后面都有注释信息，但是关键语句后面一定有注释，而且要达到言简意赅、没有歧义。四，易改易用，程序在调试时很容易修改其中的关键内容而对其它部分没有影响，当程序中需要加入内容或是某一部分需要复制到其它程序中应用时要容易修改、方便移植，虽然 C 语言本身的移植性就很好，但是在我们写程序时依然需要考虑。以上四条有哪位仁兄不同意的可以出来和我单挑，挑不过我的话我请你吃猪肉炖粉条。如果都没意见我就用实践巩固一下我的“四条方针”。注意了，现在我们开始真正动笔写 C 语言程序了，是不是有些激动呢。从程序思路到流程图，我们早已不再迷茫，调光台灯就像一座灯塔，一座可调亮度的灯塔，为我们的胜利指明了方向。

图 2 所示是我们的程序，我们要做的是把它变成具体语句。这包括初始化程序、开灯判断程序、灯泡驱动程序、关灯判断程序、关灯程序、“变亮”判断程序、亮度值增加程序、“变暗”判断程序和亮度值减少程序，一共 9 个程序块。其中包含了 5 个循环体，它们是开灯判断循环、关灯判断循环、亮度增加循环、亮度减少循环和无操作循环。寻找循环体是挺容易的，只要找准所有状态，找到箭头构成的环路就是循环体。

[程序 1]

程序名：	调光台灯
编写人：	杜洋
初写时间：	2007 年 12 月 2 日
程序功能：	三个按键控制台灯开关、变亮和变暗功能
CPU 说明：	AT89S52 型单片机 12MHz 晶体振荡器
修改日志：	N0.1-2007 年 12 月 2 日 实现基本功能

好，现在开始写程序了，在程序文本最开始的部分要加一段说明文字，介绍程序名、作者、编程时间之类的信息。如果不写是对程序没有影响，只是不附合通常的编程习惯，就像我前面说的，程序不只是能用就行的。说明文字如 [程序 1] 所示。

[程序 2]

```
#include <AT89X52.h> //库文件定义（定义单片机为 8952 系列）

sbit LED = P1 ^ 0; //台灯控制端口（VD，P1.0）
sbit ONOFF = P2 ^ 0; //开关键（SB1，P2.0）
sbit ADD = P2 ^ 1; //变亮键（SB2，P2.1）
sbit DEC = P2 ^ 2; //变暗键（SB3，P2.2）

unsigned char Brightness; //亮度值，值域 0~255（全局变量）
```



```
bit MARK; //状态标志位（目前是开灯还是关灯状态）
```

通常，说明文字下面接着就是定义信息，如[程序 2]所示。空行将其分成三段，其中第一段是定义库文件，也就是所用的单片机型号的定义，C 语言将寻找相应的库文件，了解库文件对单片机接口和一些重要内容的定义，暂时不需要了解太深。定义库文件的基本格式是“#include <库文件名>”，库文件要和你实际应用的单片机对应。第二段是接口定义部分，主要是对连接到单片机上的元件做一个对应定位，在前面闪烁 LED 程序中已经说了一些。其基本格式是“sbit 定义名称 = 接口标号;”。第三段是定义全局变量，其实就是定义了两个变量，在下面的程序中用它们记录一些重要数据，所有的函数（程序）都可以使用这两个变量，因为它们是全局变量。“unsigned char 变量名称;”是定义变量的基本格式，其变量值可以是 0 到 255 之间，而“bit 变量名称;”是定义一个位变量，它的值只能是 0 或者 1，有一些开关量用它来定义再好不过了。

注释信息是采用“//”开头的，在格式上为达到美观就应该将相同的结构对齐，使用键盘上的“Tab”键可以很快空出需要的距离，让注释在程序后面对齐排版。这是美观问题，多注意一下你的程序就会写的很漂亮。

[程序 3]

```
void Delay1mS(unsigned int a){ //延时程序(1ms)
    unsigned char i;
    while( --a != 0){
        for(i = 0; i < 125; i++);
    }
}
```

[程序 4]

```
void delay (unsigned char d) // 延时功能函数--由 d 决定延时长度
{
    unsigned char i;
    while( --d != 0)
    {
        for(i = 0; i < 2; i++);
    }
}
```

[程序 3]和[程序 4]是两个延时程序，它们的延时方式和结构都是相同的，只是延时单位时间不同，[程序 3]是用在按键去抖动时使用的，因为凡是微动开关的按键都会加一个 20 毫秒的延时，以防止按键按下瞬间不稳定时期所带来的干扰。[程序 4]是用在灯泡占空比调光上的，这要求延时单位比较小。为了给大家介绍一个概念我特意改出一个不同，那就是 int 和 char 的区别。通俗的讲这是定义数据类型的不同，unsigned int 的数据范围在 0~65535 之间，unsigned char 的数据范围在 0~255 之间，如果你的延时需要很长时间就应该用 unsigned int。上面的数据类型你也可以改一改实验看有什么变化。这两个程序在程序流程图里并没有提到是因为它们只算是辅助程序部分，这里先讲到延时程序应该是不符合初学者的逻辑，这并不是无源无故加进来的，当下面的程序用到延时程序你再回过头来看一看吧。

一个函数体的基本格式是“类型 函数名 (参数表){ 函数内容 }”。以[程序 3]为例，无返回值的、有参数表的，也就是说程序运行结束后没有需要输出的数据，而在程序运行之前需要有一些输入数据。无数据用 void 表示，有数据的地方就设置一个变量而已。在函数内部的第一句就又设置了一个变量，这是为下面的累加数延时做准备的。

下面就是延时程序了，我们一直在用它却并不了解它是怎么延时的。其实没什么了不起，就是让单片机一直做小学的算数题，要延时多久就算多少题，这段时间单片机不能干其它事情就达到延时目的，看上去很安静的单片机却一直在忙着。“while(--a != 0){ }”是一个 while 循环，当小括号中的数据为真时函数将执行大括号里的程序，“--a != 0”的意思是 a 的值减 1 后不等于 0，“--a”是指 a 的值减 1，“!=”是不等于的意思。“for(i = 0; i < 125; i++);”是和 while 函数的功能相似的，它是先设定一个变量 i 等于 0，然后判断 i 是否小于 125，如果小于则执行大括号里的程序，然后让 i 加 1，再重新判断 i 是否小于 125，直到 i 大于等于 125 的时候函数才会结束循环。细心的朋友会发现“for(i = 0; i < 125; i++);”中根本就没有大括号，其实是用“;”代替了，这相当于“for(i = 0; i < 125; i++){ }”就是执行了空指令。现在我们假设调用“Delay1mS(20);”这个函数，看看延时程序是如何完成工作的，首先 a 的值是 20，之后执行“while(--a != 0){ }”，a 的值变成 19 则不等于 0，执行大括号里的程序“for(i = 0; i < 125; i++);”，i 的值是 0 则小于 125，程序执行一次空操作，i 的值加 1 变成 1 还小于 125 再执行一次空操作，直到 i 的值大于等于 125 时程序才停止循环。不过这还不算完，外面一层的“while(--a != 0){ }”又拦住了程序，这次 a 再减 1 等于 18，还是不等于 0，又跳进了 for 循环空操作 125 次，就这样一遍又一遍的走呀走呀，终于到了 a 等于 0 的时候，这时整个延时程序宣告结束。一共执行了 20 个 125 次空操作，全部过程耗时 20mS，达到了延时目的。

大括号的位置也是美观度的一个考验，大括号是函数体的重要构成，在[程序 3]和[程序 4]中是两种常见在排版方式，其位置不会影响程序内容，较好的摆放大括号有助于让程序一目了然、简洁干净。我个人喜欢前者，因为这样节省行数，又可以方便了解程序结构。

[程序 5]

```
void drive (void){           //
    unsigned char a;         // a 控制延时长度
    a = Brightness;          //取得亮度值数据
    LED = 0;                 //点亮灯泡
    delay (a);               // 延时长度随 a 而改变
    a = ~a;                  // a 值取反  决定灯灭时的占空比
    LED = 1;
    delay (a);
    a = ~a;                  // a 值取反  使 a 回到原值继续循环
}
```

[程序 5]是灯泡的驱动程序，它将从亮度值数据中读出亮度值并根据这个值来驱动灯泡。其实关键是亮度调整的实现，我是设置了一个固定的时间长度为 255 个时间段，如果灯泡只在 255 个时间段中都是点亮的那么灯泡达到其最大亮度，如果灯泡只在 255 个时间段中的其中 1 个延时是点亮那么灯泡就是闪烁的效果，就好像前面学过的闪烁 LED 一样。幸好人体工学给了我们一个机会，当灯泡闪烁速度足够快时人眼就分辨不出闪烁而是看到亮度较低的光亮。[程序 5]就是让灯泡在亮度值的时间段点亮，再将熄灭给余下的时间段，这就是传说中的点空比。“LED = 0;”是令 LED 的值变成 0，也就是低电平。“a = ~a;”的意思是将 a 的值以 255 为准取反，如果 a 的值是 0 取反后是 255，如果 a 的值是 5 取反后是 250，这是有趣的数字游戏。

程序中必然有一些重复和类似的内容，我们看情况可以不加注释信息，例如[程序 5]中就有 2 条语句没有注释信息，可以联系上下文谁都明白这 2 个语句是何含意。为了解读程序方便是要加注释信息的，但要适量、适度。

[程序 6]

```
void open (void){                                //
    if(ONOFF == 0){                              //如果 ONOFF 为 0 则证明开关键按下
        Delay1mS(20);                          //等待 20 毫秒躲过按键不稳定的状态
        if(ONOFF == 0){                        //再看开关键是否被按下
            MARK = 1;                          //将标志位变成开灯状态
            while(ONOFF == 0);                //等待按键放开
        }
    }
}
```

[程序 7]

```
void close (void){                              //
    if(ONOFF == 0){                              //如果 ONOFF 为 0 则证明开关键按下
        Delay1mS(20);                          //等待 20 毫秒躲过按键不稳定的状态
        if(ONOFF == 0){                        //再看开关键是否被按下
            MARK = 0;                          //将标志位变成关灯状态
            while(ONOFF == 0);                //等待按键放开
        }
    }
}
```

[程序 8]

```
void add (void){                                //
    if(ADD == 0){                              //如果 ADD 为 0 则证明变亮键按下
        Brightness++;                          //亮度值加 1
        if(Brightness > 254){                  //如果亮度值大于最大值时
            Brightness = 254;                  //则保持最大值状态
        }
    }
}
```

[程序 9]

```
void dec (void){                                //
    if(DEC == 0){                              //如果 DEC 为 0 则证明变暗键按下
        Brightness--;                          //亮度值减 1
        if(Brightness < 1){                    //如果亮度值小于最小值时
            Brightness = 1;                    //则保持最小值状态
        }
    }
}
```

[程序 6]、[程序 7]、[程序 8]、[程序 9]是按键判断和处理程序，它们看上去都有相似之处。其中[程序 6]和[程序 7]又非常的相似，只有一点点不同，按照我们前面讲到的相同程序可以变成子程序的原理可不可以将[程序 6]

和[程序 7]合并成一个程序呢？是可以的，我把这么好的差史留给你来玩吧。“if(ONOFF == 0){ }”是一个判断语句，意思是如果 ONOFF 等于 0 则执行大括号里的程序，如果不等于 0 则跳过这里继续执行其他程序。值得注意的是“=”和“==”的区别，这和小学学过的数学表示不同，这种有趣的表述是 C 语言的特长。“A = B”是指将 B 的值传给 A，“A == B”的意思是 A 等于 B，多看几次实例程序自然十拿九稳。

程序的层次关系是自然存在的，总有一些程序是套在另一个程序里面的，这种一环套一环的程序结构对于熟悉汇编语言的朋友是一时半会很难适应的，在不断练习之后 C 语言的特点渐渐清楚，再回头套用就简单了。如果你现在还不怎么会套用也没关系，只要你不断的参考、分析别人的程序，自然而然的本能就会钻进你的脑袋。依然使用“Tab”键空出适合的空间让结构看上去层次分明，大括号里面的程序要比外面的多缩进一段，通过语句位置判断层次，这个习惯值得养成，与人方便自己方便。

[程序 10]

```
void init (void){           //
    LED      =  1;          //灯泡和按键初始状态设置
    ONOFF    =  1;
    ADD      =  1;
    DEC      =  1;
    Brightness = 130;        //初始化亮度值
    MARK     =  0;          //状态切换标志位
}
```

初始化程序可算是程序中必不可少的部分，包括单片机内部的初始化数据和对外部器件的初始化，就是制定出最开始的工作状态，当然也是可以将初始化的内容分散的写在程序中，不过这样做会没有条理而让日后的修改更困难。[程序 10]是将灯泡、按键、亮度值和标志位在单片机运行的最初变成这个状态。

以上所有的模块化程序讲完，主要的功能和结构都随着介绍了。这些模块化程序是依据程序流程图制作的，模块之间需要互通的数据已经在程序最开始设计成了全局变量。KEIL 在编译 C 语言程序的时间很讲究，它如果发现 A 函数中调用了 B 函数，那它一定要在 A 函数的前面见过 B 函数或是 B 函数的声明才行，所以我们尽量调整顺序到最佳状态。

[程序 11]

```
void main (void){           // 主函数 实现程序流程           (1)
    init();                  //调用初始化程序                 (2)
    while (1){               //循环结构                       (3)
        if(MARK == 0){       //状态标志为关灯状态时         (4)
            open();           //等待开关键按下（开灯）       (5)
        }
        if(MARK == 1){       //状态标志为开灯状态时         (6)
            drive();           //驱动灯泡                   (7)
            close();           //判断开关键按下（关灯）     (8)
            add();             //判断变亮键按下             (9)
            dec();             //判断变暗键按下             (10)
        }
    }
```

```
}  
}
```

每个完整的程序都会有一个 main 函数，程序将从这里开始，我们前面的所有程序都要在这里按顺序乖乖排好，这相当于结构框图中各模块间箭头的功能。在[程序 11]中，(2) 是 main 函数也就是整个程序运行的第一步，调用上面设计好的 init();初始化函数。之后进入一个无限循环体，(3) 是一种实现循环的结构，while(1){ } 中的条件判断使终是 1 也就是永远为真，这就实现了循环体里面的程序是无限循环的，在它之前的程序只在开机时被调用一次，在它之后的程序总是不会被调用。这是单片机编程的主流方法，这时的 CPU 是始终不停的工作的，如果这种 CPU 占用率是百分之百的情况出现在你的电脑上时，你可能会很害怕，这是耗费电能又让电脑速度变得超慢。我们很少在单片机程序中加入资源管理的部分，也就是没有事情可做时让 CPU 处在空闲状态，而空闲状态在 LINUX、μC/OS 等操作系统里都是在自然不过的事情。(4) 和 (6) 都是状态判断，(4) 是指当 MARK 等于 0 时则表示灯现在是关着的，进入函数里运行等待开关按下的开关程序，(6) 是指当 MARK 等于 1 时则表示灯目前是开着的，进入函数里运行灯泡驱动和判断开关键和两个调光键是否被按下，其实 MARK 的状态开关构成了两个循环。

从格式可以看到一种风格和编程态度，至少我们需要达到的应该贯穿始终。一段程序是给单片机看、给自己看、给别人看的，在最开始的日子尽量多为三方着想，发现问题并解决它，从中积累经验，回头把经验用在实践中，再发现问题。虽不是一朝一夕但也不会太久，个性的编程风格会来到你身边，助你成为具有独特风味的编程高手。

调试即将成功

程序简单调试也就简单，但二者并不总是成正比关系，有时一个小失误就会耗费许多时间和精力，不亚于恋爱中的悲欢离合。调试是编程开发的重要环节，编写程序是在为画面勾勒出大体的轮廓，而调试是细细描绘的过程。在没有仿真器的情况下，只把程序下载到单片机中看效果是比较低效率的方法，当设计一些大程序时是有一些困难。还好我们现在玩的都应该算是小程序，调光台灯也算是小程序了，仅就其为例聊一聊不用仿真器时小程序的调试方法吧。

调试的目标就是要运行正常，这包括程序中所有的运行状态，我需要做的就是模拟出各种不同的状态给程序，看它能不能输出理想的结果。回头看看调光电路有多少种状态，这是考验每个爱好者总结能力的时候，认真的态度依然要保持。我们的调试是将程序用 ISP 下载线下载到单片机里运行，然后模拟出各种状态观察程序结果的过程，这个过程可以直接在目标板上完成，也就是最终实用的调光台灯电路。我们所要总结的有正常的工作状态和异常的状态，我们并不能保证使用者一定按我们的意图操作，所以我们在调试时为了保证该实现的功能一定要能正确实现，还要站在使用者的角度去考虑人性化和安全性。虽然现在单片机爱好者都只是为了玩一玩也没有太多讲究，不过想设计出好的作品就应该考虑得更多、更细致，从这一点讲我们确实应该好好学习一下日本人的认真精神。不要摆出民族情绪哦，要理智的想到人家比我们优秀就是值得学习。

调光台灯的正常状态有以下几部分：接通电源的初始化状态、开灯操作时的状态、关灯操作时的状态、变亮操作时的状态、变暗操作时的状态。异常状态则是在正常操作步骤之外使用者别出新材的操作状态，这包括：初始化后按下“变亮”和“变暗”键时的状态、在关灯状态下按下“变亮”和“变暗”键时的状态、在开灯状态下同时按下“变亮”和“变暗”键时的状态、长按“变亮”和“变暗”键时程序最终的状态、各种状态下同时按下三个按键的状态等等。这些都是使用说明书上没有的内容，应该保证异常操作时程序无动作或是尽量驱向于用户可以接受的动作。要把出现的问题列出来看看是否需要解决和如何解决，这个过程中你就不是设计者了，而是一个纯粹的使用者，如果自己做不到也可以让其他人使用一下，因为设计作品是为了给人用的，不是高科技的陈列品。虽然人性化的工作在程序设计时就已经考虑过，但是经验告诉我在调试阶段是最容易发现问题、产生想法的时候。比如现在的调光台灯中就存在一个不是问题的问题，在开、关灯时灯光是突然点亮或熄灭的，如果设计成开灯时渐渐点亮和关灯时



渐渐熄灭可以让使用者的眼睛慢慢适应灯光，这种考虑就是实际使用时发现的，从编程技术角度看并不困难。我现在的程序里没有这个功能，有兴趣你就试试加上它吧。

总之，我想说的是设计比技术重要、认真比创新重要、思路比经验重要、借鉴比原创重要、实践比理论重要、人性化比高、精、尖重要、专注细节比功能强大重要、快乐比成功重要。也许你不同意我的观点，没关系，玩得开心就好！

版本信息

题目	单片机编程魔法学校
作者	杜 洋
时间	2008.3.1
版本	V1.0
声明	本站内容（包括程序代码、文档、照片、视频等）属个人所有，未经网站作者同意请勿转载或引用，对于转载或复制而造成的任何不良后果概不负责。 对于本站内免费下载的资料、图片及视频不能保证其真实可靠，对于免费下载的程序代码本站作者不给予技术支持和服务。

本文版权属《无线电》杂志所有，DoYoung.net 经特许转载！