手把手教你学单片机的C语言程序设计(六)

编译预处理

◆周兴华

所谓编译预处理,是编译器在对C语言源程序进行正常编译之前,先对一些特殊的预处理命令作解释,产生一个新的源程序。编译预处理主要为程序调试、移植等提供便利,是一个非常实用的功能。

宏定义

在源程序中,为了区分预处理命令和一般的C语句的不同,所有预处理命令行都以符号"#"开头,并且结尾不用分号。预处理命令可以出现在程序任何位置,但习惯上尽可能地写在源程序的开头,其作用范围从其出现的位置到文件尾。

C语言提供的预处理命令主要有:宏定义、文件包含和条件编译。其中宏定义分为带参数的宏定义和不带参数的宏定义。

1.不带参数的宏定义

不带参数的宏定义的一般形式为:

#define 标识符 字符串

它的作用是在编译预处理时,将源程序中所有标识符替换成字符串。例如:#define PI 3.14

#define uint unsigned int

当需要修改元素时,只要直接修改宏定义即可,无需修改程序中所有出现元素个数的地方。所以宏定义,不仅提高了程序的可读性、便于调试,而且也方便了程序的移植。

无参数的宏定义使用时,要注意以下几个问题:

1).宏名一般用大写字母,以便于与变量名的区别。当然,用小写字母也不为错。

2).在编译预处理中宏名与字符串进行替换时,不作语法检查,只是简单的字符替换,只有在编译时才对已经展开宏名的源程序进行语法检查。

3).宏名的有效范围是从定义位置到文件结束。如果需要终止宏定义的作用域,可以用#undef命令。例如:

#undef PI

则该语句之后的PI不再代表3.14,这样可以灵活控制宏定义的范围。

4).宏定义时可以引用已经定义的宏名。例如:

#define R 2.0

#define PI 3.14

#define ALL PI*R

5).对程序中用双引号扩起来的字符串内的字符,不进行宏的替换操作。

2.带参数的宏定义

为了进一步扩大宏的应用范围,在定义宏时,还可以带参数。带参数的宏定义的一般形式为:

#define 标识符(参数表) 字符串

它的作用是在编译预处理时,将源程序中所有标识符替换成字符串,并且将字符串中的参数用实际使用的参数替换。例如:

#define S(a,b) (a+b)/2

则源程序中如果使用了S(3,4),在编译预处理时将替换为(3+4)/2。

实验一

在LED/16*2字符液晶试验板上实现两数相加并输出结果,变量的数据类型用宏定义的缩写形式。

在我的文档中建立一个文件目录(cs10),然后建立cs10.uv2的工程项目,最后建立源程序文件(cs10.c)。

输入下面的程序:

```
#include <REG51.H>// 序号(以下同):1
#define uchar unsigned char  //2
uchar code SEG7 [10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,};//3
//=================4=======
void main(void)                    //5
{                                  //6
    uchar a,b,sum;                 //7
    a=55;                          //8
    b=200;                         //9
    sum=a+b;                       //10
    P2= SEG7[ sum/100];            //11
    P1= SEG7[ (sum%100)/10];       //12
    P0= SEG7[ sum%10];             //13
    while(1);                      //14
}                                  //15
```

编译通过后,将生成的cs10.hex文件烧录到89S51芯片中,将芯片插入到LED/16*2字符液晶试验板上,试验板上接通9V电源,右边3个LED数码管显示"255"。通过宏定义,我们发现原来长长的"unsigned char"现变成了"uchar",是不是更方便使用了。

分析程序。

序号1(程序解释,以下同):包含头文件REG51.H。

序号2:数据类型"unsigned char"用宏定义为简写形式"uchar"。

序号3:数码管0~9的字形码。

序号4:程序分隔。

序号5:定义函数名为main的主函数。

序号6:main的主函数开始。

序号7:定义无符号字符型变量a、b、sum。

序号8:a赋值55。

序号9:b赋值200。

序号10:a、b作加法运算,其和放sum。

序号11:取出sum的百位数送P2口显示。

序号12:取出sum的十位数送P1口显示。

序号13:取出sum的个位数送P0口显示。

序号 14:动态停机。
序号 15:main 的主函数结束。

实验三

使用带参数的宏定义进行运算,结果送 LED/16*2 字符液晶试验板显示。

在我的文档中建立一个文件目录(cs11),然后建立 cs11.uv2 的工程项目,最后建立源程序文件(cs11.c)。

输入下面的程序:

```
#include <REG51.H>// 序号(以下同):1
#define uchar unsigned char  //2
#define S(a,b) (a-b)*3       //3
uchar code SEG7 [10]=
{0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf
8,0x80,0x90,};//4
//==================5=======
void main(void)              //6
{                            //7
uchar out;                   //8
out= S(100,50);              //9
    P2= SEG7[ out/100];      //10
    P1= SEG7[ (out%100)/10];//11
    P0= SEG7[ out%10];       //12
    while(1);                //13
}                            //14
```

编译通过后,将生成的 cs11.hex 文件烧录到 89S51 芯片中,将芯片插入到 LED/16*2 字符液晶试验板上,试验板上接通 9V 电源,右边 3 个 LED 数码管显示"150"。

分析程序。

序号 1(程序解释,以下同):包含头文件 REG51.H。
序号 2:数据类型的宏定义。
序号 3:带参数的宏定义
序号 4:数码管 0~9 的字形码。
序号 5:程序分隔。
序号 6:定义函数名为 main 的主函数。
序号 7:main 的主函数开始。
序号 8:定义无符号字符型变量 out。
序号 9:使用参数宏,其结果送 out。
序号 10:取出 out 的百位数送 P2 口显示。
序号 11:取出 out 的十位数送 P1 口显示。
序号 12:取出 out 的个位数送 P0 口显示。
序号 13:动态停机。
序号 14:main 的主函数结束。

文件包含

"文件包含"实际上就是我们前面已经多次用到的 #include 命令实现的功能,即一个源程序文件可以包含另外一个源程序文件的全部内容。"文件包含"不仅可以包含头文件,例如:#include <REG51.H>,还可以包含用户自己编写的源程序文件,例如:#include <MY_PROG.C>。

文件包含预处理命令的一般形式为:

#include < 文件名 > 或 #include "文件名"

上述两种方式的区别是:前一种形式的文件名用尖括弧括起来,系统将到包含 C 语言库函数的头文件所在的目录(通常是 KEIL 目录中的 include 子目录)中寻找文件。后一种形式的文件名用双引号括起来,系统先在当前目录下寻找,若找不到,再到其它路径中查找。

"文件包含"在使用时要注意:

1. 一个 #include 命令只能指定一个被包含的文件。

2. "文件包含"可以嵌套。在文件包含的嵌套时,如果文件 1 包含了文件 2,而文件 2 包含了文件 3,则在文件 1 也要包含文件 3,并且文件 3 的包含要写在文件 2 的包含之前,即文件 1 中的"文件包含"说明如下:

```
#include < 文件名 1>
#include < 文件名 2>
```

"文件包含"命令为多个源程序文件的组装提供了一种方法。在编写程序时,习惯上将公共的符号常量定义、数据类型定义和 extern 类型的全局变量说明构成一个源文件,并以".H"为文件名的后缀。如果其他文件用到这些说明时,只要包含该文件即可,无需再重新说明,减少了工作量。而且这样编程使得各源程序文件中的数据结构、符号常量以及全局变量形式统一,便于程序的修改和调试。

条件编译

"条件编译"命令允许对程序中的内容选择性地编译,即可以根据一定的条件选择是否编译。

条件编译的命令主要有以下几种形式:

形式 1.

```
#ifdef 标识符
程序段 1
#else
程序段 2
#endif
```

它的作用是当"标识符"已经由 #define 定义过了,则编译"程序段 1",否则编译"程序段 2"。其中如果不需要编译"程序段 2",则上述形式可以变换为:

```
#ifdef 标识符
程序段 1
#endif
```

形式 2.

```
#ifndef 标识符
程序段 1
#else
程序段 2
#endif
```

它的作用是当"标识符"没有由 #define 定义过,则编译"程序段 1",否则编译"程序段 2"。同样当无"程序段 2"时,则上述形式变换为:

```
#ifndef 标识符
程序段 1
#endif
```

形式 3.

```
#if 表达式
程序段 1
#elsee
程序段 2
#endif
```

它的作用是当"表达式"值为真时,编译程序段 1,否则编译程序段 2。同样当无程序段 2 时,则上述形式变换为:

```
#if 表达式
程序段 1
```

#endif

以上三种形式的条件编译预处理结构都可以嵌套使用。当 #else 后嵌套 #if 时，可以使用预处理命令 #elif，它相当于 #else #if。

在程序中使用条件编译主要是为了方便程序的调试和移植。

重新定义数据类型

在 C 语言程序中，用户可以根据自己的需要对数据类型重新定义。使用关键字 typedef 的定义方法如下：

typedef 已有的数据类型新的数据类型名;

其中“已有的数据类型”是指上面所介绍的 C 语言中所有的数据类型，包括结构、指针和数组等，“新的数据类型名”可按用户自己的习惯或根据任务需要决定。关键字 typedef 的作用只是将 C 语言中已有的数据类型作了置换，因此可用置换后的新数据类型名来进行变量的定义。例如：

typedef int WORD; // 定义 word 为新的整型数据类型名

一般而言，对 typedef 定义的新数据类型用大写字母表示，以便与 C 语言中原有的数据类型相区别。另外还要注意，用 typedef 可以定义各种新的数据类型名，但不能直接用来定义变量。typedef 只是对已有的数据类型作了一个名字上的置换，并没有创造出一个新的数据类型，例如前面例子中的 WORD，它只是 int 类型的一个新名字而已。采用 typedef 来重新定义数据类型有利于程序的移植，同时还可以简化较长的数据类型定义（如结构数据类型等）。在采用多模块程序设计时，如果不同的模块程序源文件中用到同一类型的数据时（尤其是像数组、指针、结构、联合等复杂数据类型），经常用 typedef 将这些数据重新定义并放到一个单独的文件中，需要时再用预处理命令 #include 将它们包含进来。

实验三

用 typedef 重新定义数据类型，变量 val 赋值后送 LED/128*64 图形液晶试验板上的 8 个 LED 显示，模拟彩灯闪烁。

在我的文档中建立一个文件目录(cs12)，然后建立 cs12.uv2 的工程项目，最后建立源程序文件(cs12.c)。输入下面的程序：

```
#include <REG51.H>// 序号(以下同):1
typedef unsigned char U8_BYTE;   //2
typedef unsigned int U16_WORD;   //3
void delay(U16_WORD k);          //4
//==============5==========
void main(void)                  //6
{                                //7
U8_BYTE val;                     //8
    while(1)                     //9
    {                            //10
val=0x55;                        //11
P0=val;                          //12
delay(500);                      //13
val=0xaa;                        //14
P0=val;                          //15
delay(500);                      //16
    }                            //17
}                                //18
//---------------19--------
void delay(U16_WORD k)           //20
{                                //21
U16_WORD i,j;                    //22
for(i=0;i<k;i++){                //23
for(j=0;j<121;j++)               //24
{;}}                             //25
}                                //26
```

编译通过后，将生成的 cs12.hex 文件烧录到 89S51 芯片中，将芯片插入到 LED/128*64 图形液晶试验板上，拔下 SX 排针上的 8 个短路块。试验板接通 5V 电源后，8 个 LED 的奇数位点亮 0.5 秒，然后偶数位又点亮 0.5 秒，反复循环，煞是好看。

我们对程序进行分析。

序号 1（程序解释，以下同）：包含头文件 REG51.H。

序号 2：用 typedef 重新定义数据类型，定义 U8_BYTE 为 8 位长度的无符号字符型数据类型。

序号 3：用 typedef 重新定义数据类型，定义 U16_WORD 为 16 位长度的无符号整型数据类型。

序号 4：延时子函数声明。

序号 5：程序分隔。

序号 6：定义函数名为 main 的主函数。

序号 7：main 的主函数开始。

序号 8：定义无符号字符型变量 val。

序号 9：while 循环语句进行无限循环。

序号 10：while 循环语句开始。

序号 11：val 赋值 0x55。

序号 12：val 送 P0 口显示。

序号 13：调用 0.5 秒延时子函数。

序号 14：val 赋值 0xaa。

序号 15：val 送 P0 口显示。

序号 16：调用 0.5 秒延时子函数。

序号 17：while 循环语句结束。

序号 18：main 的主函数结束。

序号 19：程序分隔。

序号 20~26：延时子函数。