手把手教你学单片机的C语言程序设计(十五)

◆吕超亚

结构体、共用体及枚举

前面我们介绍了C语言的基本数据类型,但是在实际设计一个较复杂程序时,仅有这些基本类型的数据是不够的,有时需要将一批各种类型的数据放在一起使用,从而引入了所谓构造类型的数据,例如前面介绍的数组就是一种构造类型的数据,一个数组实际上是将一批相同类型的数据顺序存放。这里我们还要介绍C语言中另一类更为常用的构造类型数据:结构体、共用体及枚举。

结构体的概念

结构体是一种构造类型的数据,它是将若干个不同类型的数据变量有序地组合在一起而形成的一种数据的集合体。组成该集合体的各个数据变量称为结构成员,整个集合体使用一个单独的结构变量名。一般来说结构中的各个变量之间是存在某些关系的,例如时间数据中的时、分、秒,日期数据中的年、月、日等。由于结构是将一组相关联的数据变量作为一个整体来进行处理,因此在程序中使用结构将有利于对一些复杂而又具有内在联系的数据进行有效的管理。

结构体类型变量的定义

1.先定义结构体类型再定义变量名

定义结构体类型的一般格式为:

```
struct 结构体名
  {
成员表列
};
```

其中,"结构体名"用作结构体类型的标志。"成员表外"为该结构体中的各个成员,由于结构体可以由不同类型的数据组成,因此对结构体中的各个成员都要进行类型说明。

例如定义一个日期结构体类型date,它可由6个结构体成员year、month、day、hour、min、sec组成:

```
struct date
{
int year;
char month;
char day;
char hour;
char min;
char sec;
};
```

定义好一个结构体类型之后,就可以用它来定义结构体变量。一般格式为:

struct 结构体名 结构体变量名1,结构体变量名2,……结构体变量名n;

例如可以用结构体date来定义两个结构体变量time1和time2:

```
struct date time1,time2;
```

这样结构体变量time1和time2都具有struct date类型的结构,即它们都是1个整型数据和5个字符型数据所组成。

2. 在定义结构体类型的同时定义结构体变量名

一般格式为:

```
struct 结构体名
  {
成员表列
} 结构体变量名1,结构体变量名2,……结构体变量名n;
```

例如对于上述日期结构体变量也可按以下格式定义:

```
struct date
{
int year;
char month;
char day;
char hour;
char min;
char sec;
}time1,time2;
```

3.直接定义结构体变量

一般格式为:

```
struct
  {
成员表列
} 结构体变量名1,结构体变量名2,……结构体变量名n;
```

第3种方法与第2种方法十分相似,所不同的只是第3种方法中省略了结构体名。这种方法一般只用于定义几个确定的结构变量的场合。例如,如果只需要定义time1和time2而不打算再定义任何别的结构变量,则可省略掉结构体名"tate"。

不过为了便于记忆和以备将来进一步定义其他结构体变量的需要,一般还是不要省略结构名为好。

关于结构体类型有几点需要注意的地方:

1. 结构体类型与结构体变量是两个不同的概念。定义一个结构体类型时只是给出了该结构体的组织形式,并没有给出具体的组织成员。因此结构体名不占用任何存储空间,也不能对一个结构体名进行赋值、存取和运算。

而结构体变量则是一个结构体中的具体对象,编译器会给具体的结构体变量名分配确定的存储空间,因此可以对结构体变量名进行赋值、存取和运算。

2. 将一个变量定义为标准类型与定义为结构体类型有所不同。前者只需要用类型说明符指出变量的类型即可,如int x;。后者不仅要求用struct指出该变量为结构体类型,而且还要求指出该变量是哪种特定的结构类型,即要指出它所属的特定结构类型的名字。如上面的date就是这种特定的结构体类型(日期结构体类型)的名字。

3. 一个结构体中的成员还可以是另外一个结构体类型的变量，即可以形成结构体的嵌套。

结构体变量的引用

定义了一个结构体变量之后，就可以对它进行引用，即可以进行赋值、存取和运算。一般情况下，结构体变量的引用是通过对其成员的引用来实现的。

1. 引用结构体变量中的成员的一般格式为：

结构体变量名. 成员名

其中“.”是存取成员的运算符。

例如：time1.year=2006;表示将整数 2006 赋给 time1 变量中的成员 year。

2. 如果一个结构体变量中的成员又是另外一个结构体变量，即出现结构体的嵌套时，则需要采用若干个成员运算符，一级一级地找到最低一级的成员，而且只能对这个最低级的结构元素进行存取访问。

3. 对结构体变量中的各个成员可以像普通变量一样进行赋值、存取和运算。

例如：time2.sec++;

4. 可以在程序中直接引用结构体变量和结构体成员的地址。结构体变量的地址通常用作函数参数，用来传递结构体的地址。

结构体变量的初始化

和其它类型的变量一样，对结构体类型的变量也可以在定义时赋初值进行初始化。

例如：

```
struct date
{
int year;
char month;
char day;
char hour;
char min;
char sec;
}time1={2006,7,23,11,4,20};
```

结构体数组

一个结构体变量可以存放一组数据(如一个时间点 time1 的数据)，在实际使用中，结构体变量往往不止一个(例如我们要对 20 个时间点的数据进行处理)，这时可将多个相同的结构体组成一个数组，这就是结构体数组。

结构体数组的定义方法与结构体变量完全一致，例如：

```
struct date
{
int year;
char month;
char day;
char hour;
char min;
char sec;
};
struct date time[20];
```

这就定义了一个包含有 20 个元素的结构体数组变量 time，其中每个元素都是具有 date 结构体类型的变量。

指向结构体类型数据的指针

一个结构体变量的指针，就是该变量在内存中的首地址。我们可以设一个指针变量，将它指向一个结构体变量，则该指针变量的值是它所指向的结构体变量的起始地址。

定义指向结构体变量的指针的一般格式为：

```
struct 结构体类型名 * 指针变量名
或
struct
{
成员表列
} * 指针变量名;
```

与一般指针相同，对于指向结构体变量的指针也必须先赋值后才能引用。用指向结构体变量的指针引用结构体成员

通过指针来引用结构体成员的一般格式为：

```
指针变量名 -> 结构体成员
```

例如：

```
struct date
{
int year;
char month;
char day;
char hour;
char min;
char sec;
};
struct date time1;
struct date *p;
p=&time1;
p->year=2006;
```

指向结构体数组的指针

我们已经了解了，一个指针变量可以指向数组。同样，指针变量也可以指向结构体数组。

指向结构体数组的指针变量的一般格式为：

struct 结构体数组名 * 指针变量名;

将结构体变量和指向结构体的指针作函数参数

结构体既可作为函数的参数，也可作为函数的返回值。当结构体被用作函数的参数时，其用法与普通变量作为实际参数传递一样，属于“传值”方式。

但当一个结构体较大时，若将该结构体作为函数的参数，由于参数传递采用值传递方式，需要较大的存储空间(堆栈)来将所有的成员压栈和出栈，此外还影响程序的执行速度。

这时我们可以用指向结构体的指针来作为函数的参数，此时参数的传递是按地址传递方式进行的。由于采用的是“传址”方式，只需要传递一个地址值。与前者相比，大大节省了存储空间，同时还加快了程序的执行速度。其缺点是在调用函数时对指针所作的任何变动都会影响到原来的结构体变量。

共用体的概念

结构体变量占用的内存空间大小是其各成员所占长度的总和，如同一时刻只存放其中的一个成员数据，对内存空间是很大的浪费。共用体也是C语言中一种构造类型的数据结构，它所占内存空间的长度是其中最长的成员长度。各个成员的数据类型及长度虽然可能都不同，但都从同一个地址开始存放，即采用了所谓的"覆盖技术"。这种技术可使不同的变量分时使用同一个内存空间，有效提高了内存的利用效率。

共用体类型变量的定义

共用体类型变量的定义方式与结构体类型变量的定义相似，也有3种方法。

1.先定义共用体类型再定义变量名

定义共用体类型的一般格式为：

```
union 共用体名
{
成员表列
};
```

定义好一个共用体类型之后，就可以用它来定义共用体变量。一般格式为：

union 共用体名 共用体变量名1，共用体变量名2，……共用体变量名n;

2. 在定义共用体类型的同时定义共用体变量名

一般格式为：

```
union 共用体名
{
成员表列
} 共用体变量名1，共用体变量名2，……共用体变量名n;
```

3.直接定义共用体变量

一般格式为：

```
union
{
成员表列
}共用体变量名1，共用体变量名2，……共用体变量名n;
```

可见，共用体类型与结构体类型的定义方法是很相似的，只是将关键字struct改成了union，但是在内存的分配上两者却有着本质的区别。结构体变量所占用的内存长度是其中各个元素所占用内存长度的总和，而共用体变量所占用的内存长度是其中最长的成员长度。

例如：

```
struct exmp1
{
int a;
char b;
};
```

struct exmp1 x; 结构体变量x所占用的内存长度是成员a、b长度的总和，a占用2字节，b占用1字节，总共占用3字节。

再如：

```
union exmp2
{
int a;
char b;
};
```

union exmp2 y; 共用体变量y所占用的内存长度是最长的成员a的长度，a占用2字节，故总共占用2字节。共用体变量的引用与结构体变量类似，对共用体变量的引用也是通过对其成员的引用来实现的。引用共用体变量的成员的一般格式为：

共用体变量名.共用体成员

枚举类型

如果一个变量只有几种可能的值，那么可以定义为枚举类型。所谓"枚举"是将变量的值一一列举出来，变量的取值只限于列出的范围内。

一个完整的枚举定义说明语句的一般格式是：

enum 枚举名{枚举值列表}变量列表;

定义和说明也可以分成两句完成：

```
enum 枚举名{枚举值列表};
enum 枚举名 变量列表;
```

例如，每星期的天数（变量weekday）只能是星期天、星期一～星期六这几种，因此可这样定义枚举变量：

```
enum weekday {sun,mon,tue,wed,thu,fri,sat}date1,date2;
```

或

```
enum weekday {sun,mon,tue,wed,thu,fri,sat};
enum weekday date1,date2;
```

说明：

1.在C编译器中，对枚举元素按常量处理，故称枚举常量。注意，不能对枚举元素进行赋值。

2. 枚举元素作为常量，它是有值的，C语言编译时按定义时的顺序使它们的值为0,1,2,……。

下面我们做实验。

实验一

在LED/128*64图形液晶试验板上，设计一个连续的计时器（对时、分、秒累计，计秒的范围为0~60，计分的范围为0~60，计时的范围为0~9999）。由于数码管只有4位，而我们显示的时（4位）、分（2位）、秒（2位）共有8位，因此采用分时显示的方法，即1分钟到时显示时，下一个1分钟到时显示分、秒，反复循环进行。我们采用计时单元和显示缓冲单元分开的方法，每次定时器中断时，时间递增，同时将时间的数值转存到显示缓冲区。我们将计时单元hour、min、sec、cnt定义成全局变量，而将显示缓冲区定义成一个日期结构体类型变量进行实验，大家可从中熟悉对结构体变量成员的操作。

在我的文档中建立一个文件目录（cs40），然后建立cs40.uv2的工程项目，最后建立源程序文件（cs40.c）。

输入下面的程序：

```
#include<REG51.H>// 序号（以下同）:1
#define uint unsigned int          //2
#define uchar unsigned char        //3
//*****************************************4
uchar code SEG7[10]={0x3f,0x06,0x5b,
0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};//5
uchar code ACT[4]={0xef,0xdf,0xbf,0x7f};//6
uint hour;                         //7
uchar min,sec,cnt;                 //8
uchar status;                      //9
void delay(uint k);                //10
//*****************************************11
struct deda                        //12
{                                  //13
uint dhour;                        //14
uchar dmin;                        //15
uchar dsec;                        //16
};                                 //17
struct deda dis_buff;              //18
//*****************************************19
void initial(void)                 //20
```

```
{                                          //21
TMOD=0x01;                                 //22
TH0=-(50000/256);                          //23
TL0=-(50000%256);                          //24
    ET0=1;                                 //25
    TR0=1;                                 //26
    EA=1;                                  //27
}                                          //28
//*****************************************29
void time0(void) interrupt 1               //30
{                                          //31
TH0=-(50000/256);                          //32
TL0=-(50000%256);                          //33
cnt++;                                     //34
if(cnt>=20){sec++;cnt=0;}                  //35
if(sec>=60){min++;sec=0;status++;}//36
if(min>=60){hour++;min=0;}                 //37
if(hour>9999){hour=0;}                     //38
if(status>=2){status=0;}                   //39
dis_buff.dhour=hour;                       //40
dis_buff.dmin=min;                         //41
dis_buff.dsec=sec;                         //42
}                                          //43
//*****************************************44
void main(void)                            //45
{                                          //46
    initial();                             //47
    for(;;)                                //48
    {                                      //49
        switch(status)                     //50
        {                                  //51
    case 0: {P0=SEG7 [dis_buff.dhour%
10];//52
    P2=ACT[0];                             //53
    delay(1);                              //54
 P0=SEG7[(dis_buff.dhour%100)/10];//55
    P2=ACT[1];                             //56
    delay(1);                              //57
    P0=SEG7  [(dis_buff.dhour% 1000)
/100];//58
    P2=ACT[2];                             //59
    delay(1);                              //60
    P0=SEG7[dis_buff.dhour/1000];//61
    P2=ACT[3];                             //62
    delay(1);}break;                       //63
    case 1: {P0=SEG7 [dis_buff.dsec%
10];//64
    P2=ACT[0];                             //65
    delay(1);                              //66
    P0=SEG7[dis_buff.dsec/10];//67
    P2=ACT[1];                             //68
    delay(1);                              //69
    P0=SEG7[dis_buff.dmin%10];//70
    P2=ACT[2];                             //71
    delay(1);                              //72
    P0=SEG7[dis_buff.dmin/10];//73
    P2=ACT[3];                             //74
    delay(1);}break;                       //75
    default:break;                         //76
        }                                  //77
    }                                      //78
}                                          //79
//*****************************************80
void delay(uint k)                         //81
{                                          //82
uint data i,j;                             //83
for(i=0;i<k;i++){                          //84
for(j=0;j<60;j++)                          //85
{;}}                                       //86
}                                          //87
```

编译通过后，将生成的 cs40.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，试验板上接通 5V 开关稳压电源。4 个 LED 数码管显示时“0000”。约 1 分钟后，4 个 LED 数码管显示分、秒“01XX”，其中秒位在不停地变化。再过约 1 分钟后，4 个 LED 数码管又显示时“0000”。……

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、3：数据类型的宏定义。

序号 4：程序分隔。

序号 5：定义数组 SEG7[10]，存放数码管 0~9 的字形码。

序号 6：定义数组 ACT4[4]，存放四位数码管的位码。

序号 7：定义“小时”变量 hour，由于累计的小时最高为 9999，因此将其定义为无符号整型变量。

序号 8：定义“分”变量 min、“秒”变量 sec、及毫秒级变量 cnt，它们均为无符号字符型变量。

序号 9：定义显示状态 status，为无符号字符型变量。

序号 10：延时子函数声明。

序号 11：程序分隔。

序号 12：定义一个日期 deda 的结构体类型。

序号 13：结构体类型开始。

序号 14：定义无符号整型变量 dhour 为成员，作为显示“时”的值。

序号 15：定义无符号字符型变量 dmin 为成员，作为显示“分”的值。

序号 16：定义无符号字符型变量 dsec 为成员，作为显示“秒”的值。

序号 17：结构体类型定义结束。

序号 18：定义 deda 类型结构体的变量 dis_buff。

序号 19：程序分隔。

序号 20：定义函数名为 initial 的初始化子函数。

序号 21：initial 子函数开始。

序号 22：定时器 T0 方式 1。

序号 23~24：T0 赋定时初值。LED/128*64 图形液晶试验板上的晶振频率为 11.0592MHz，为方便解释，我们可近似看作为 12.000MHz，这样取上面的定时初值时，T0 的定时长度近似为 50ms。

序号 25：T0 中断使能。

序号 26：启动 T0。

序号 27：开 CPU 中断。

序号 28：initial 子函数结束。

序号 29：程序分隔。

序号 30：定时器 T0 的定时中断服务子函数。

序号 31：定时中断服务子函数开始。

序号 32~33：重载定时初值。

序号 34：计数器 cnt 累加。

序号 35：计数 20 次后，恰好为 1 秒（20x50ms），这时秒单元 sec 累加，而 cnt 清除。

序号 36：秒单元 sec 计 60 次后，分单元 min 累加，而 sec 清除。同时显示状态 status 累加。

序号 37：分单元 min 计 60 次后，时单元 hour 累加，而 min 清除。

序号 38：时单元计数到 9999 后，又回到 0。

序号 39：显示状态的变化范围为 0~1。

序号 40：将时 hour 赋给结构体的变量成员 dis_buff.dhour。

序号 41：将分 min 赋给结构体的变量成员 dis_buff.dmin。

序号 42：将秒 sec 赋给结构体的变量成员 dis_buff.dsec。

序号 43：定时中断服务子函数结束。

序号 44：程序分隔。

序号 45：定义函数名为 main 的主函数。

序号 46：main 主函数开始。

序号 47：调用初始化子函数。

序号 48：for 语句用作无限循环。

序号 49：无限循环开始。

序号 50：switch 语句，根据显示状态 status 的值进行散转。

序号 51：switch 语句开始。

序号 52：当显示状态 status 为 0 时，结构变量的成员 dhous 取其个位后，再查出字形码，然后送 P0 口。

序号 53：点亮数码管的个位。

序号 54：延时 1ms 便于观察。

序号 55：结构变量的成员 dhous 取其十位后，再查出字形码，然后送 P0 口。

序号 56：点亮数码管的十位。

序号 57：延时 1ms 便于观察。

序号 58：结构变量的成员 dhous 取其百位后，再查出字形码，然后送 P0 口。

序号 59：点亮数码管的百位。

序号 60：延时 1ms 便于观察。

序号 61：结构变量的成员 dhous 取其千位后，再查出字形码，然后送 P0 口。

序号 62：点亮数码管的千位。

序号 63:延时 1ms 便于观察,然后退出。
序号 64:当显示状态 status 为 1 时,结构变量的成员 dsec 取其个位后,再查出字形码,然后送 P0 口。
序号 65:点亮数码管的个位。
序号 66:延时 1ms 便于观察。
序号 67: 结构变量的成员 dsec 取其十位后,再查出字形码,然后送 P0 口。
序号 68:点亮数码管的十位。
序号 69:延时 1ms 便于观察。
序号 70: 结构变量的成员 dmin 取其个位后,再查出字形码,然后送 P0 口。
序号 71:点亮数码管的百位。
序号 72:延时 1ms 便于观察。
序号 73: 结构变量的成员 dmin 取其十位后,再查出字形码,然后送 P0 口。
序号 74:点亮数码管的千位。
序号 75:延时 1ms 便于观察,然后退出。
序号 76:如果 status 不符合 0 或 1,则退出。
序号 77:switch 语句结束。
序号 78:for 无限循环语句结束。
序号 79:main 主函数结束。
序号 80:程序分隔。
序号 81~87:延时子程序。

实验三

实验一是在 LED/128*64 图形液晶试验板上, 设计一个连续的计时器(对时、分、秒累计),将显示缓冲区定义成一个日期结构体类型变量,这样任何时候,我们都可以同时取用显示缓冲区内的任何数据。例如:第 1 分钟显示时,而下 1 分钟同时显示分、秒,反复循环。当然,如果数码管有 8 位,那我们同时显示时、分、秒也没问题。

但是,有时我们设计的某些仪表或控制系统, 并不需要同时取用数据,而只需分时取用,这时我们没有必要将数据缓冲区定义为结构体类型,而只需定义为共用体类型即可,这将大大减少内存的开销。例如,我们将实验一的显示方式作一下修改, 第 1 分钟显示时,而下 1 分钟显示分,再下 1 分钟显示秒,反复循环。

实验三程序如下,读者可参考实验一自行分析。

在我的文档中建立一个文件目录(cs42),然后建立 cs42.uv2 的工程项目,最后建立源程序文件(cs42.c)。

输入下面的程序:

```
#include<REG51.H>// 序号(以下同):1
#define uint unsigned int                    //2
#define uchar unsigned char                  //3
uchar code SEG7[10]={0x3f,0x06,0x5b,
0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};//4
uchar code ACT[4]={0xef,0xdf,0xbf,0x7f};
                                             //5
uint hour;                                   //6
uchar min,sec,cnt;                           //7
uchar status;                                //8
void delay(uint k);                          //9
//*************************************10
union deda                                   //11
{                                            //12
uint dhour;                                  //13
uchar dmin;                                  //14
uchar dsec;                                  //15
};                                           //16
union deda dis_buff;                         //17
//*************************************18
void initial(void)                           //19
{                                            //20
TMOD=0x01;                                   //21
TH0=-(50000/256);                            //22
TL0=-(50000%256);                            //23
    ET0=1;                                   //24
    TR0=1;                                   //25
    EA=1;                                    //26
}                                            //27
//*************************************28
void time0(void) interrupt 1                 //29
{                                            //30
TH0=-(50000/256);                            //31
TL0=-(50000%256);                            //32
cnt++;                                       //33
if(cnt>=20){sec++;cnt=0;}                    //34
if(sec>=60){min++;sec=0;status++;}//35
if(min>=60){hour++;min=0;}                   //36
if(hour>9999){hour=0;}                       //37
if(status>2){status=0;}                      //38
    switch(status)                           //39
    {                                        //40
 case 0:dis_buff.dhour=hour;break;//41
    case 1:dis_buff.dmin=min;break;//42
    case 2:dis_buff.dsec=sec;break;//43
    default:break;                           //44
    }                                        //45
}                                            //46
//*************************************47
void main(void)                              //48
{                                            //49
    initial();                               //50
    for(;;)                                  //51
    {                                        //52
        switch(status)                       //53
        {                                    //54
    case 0: {P0=SEG7 [dis_buff.dhour%
10];P2=ACT[0];delay(1);                      //55
    P0=SEG7[(dis_buff.dhour%100)/10];
P2=ACT[1];delay(1);                          //56
    P0=SEG7 [(dis_buff.dhour% 1000)
/100];P2=ACT[2];delay(1);                    //57
    P0=SEG7 [dis_buff.dhour/1000];
P2=ACT[3];delay(1);}break;                   //58
    case 1: {P0=SEG7 [dis_buff.dmin%
10];P2=ACT[2];delay(1);                      //59
    P0=SEG7[dis_buff.dmin/10];P2=ACT
[3];delay(1);}break;//60
    case 2: {P0=SEG7 [dis_buff.dsec%
10];P2=ACT[0];delay(1);                      //61
    P0=SEG7[dis_buff.dsec/10];P2=ACT
[1];delay(1);}break;                         //62
        default:break;                       //63
        }                                    //64
    }                                        //65
}                                            //66
//*************************************67
void delay(uint k)                           //68
{                                            //69
uint data i,j;                               //70
for(i=0;i<k;i++){                            //71
for(j=0;j<60;j++)                            //72
{;}}                                         //73
}                                            //74
```

由于本刊篇幅所限,所列实验二及实验四移至本刊网上,有兴趣的读者可到网上查阅。