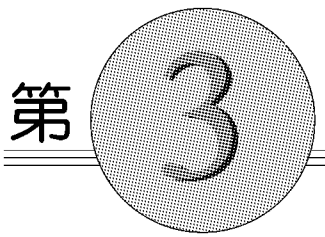

第3章 嵌入式C语言程序设计基础

本章介绍嵌入式C语言程序设计的一些基本概念。主要内容包括以下几方面。

- 嵌入式C语言预处理伪指令
- 嵌入式C语言的基本数据类型
- 嵌入式C语言程序结构
- 嵌入式C语言函数
- 嵌入式C语言数组、指针、结构体和联合

3.1 嵌入式C语言预处理伪指令

在C语言程序中常常加入一些“预处理命令”，可以改进程序设计的环境，提高编程效率。它虽然写在源程序中，但不产生程序代码，因此也称为预处理伪指令。它不是C语言本身的组成部分，因此不能直接进行编译，而必须在编译前预先对这些特殊的命令进行“预处理”。在预处理引用“预处理命令”定义的实际内容代替该命令，因此，也称为“编译预处理命令”或“编译预处理伪指令”。

C语言所有预处理伪指令行都以#开头，以区别于源文件中的语句行与说明行。预处理伪指令有以下3种：文件包含、宏定义和条件编译。

1. 文件包含伪指令

文件包含伪指令可将头文件包含到程序中，头文件中定义的内容有符号常量、复合变量原型、用户定义的变量原型和函数的原型说明等。编译器编译预处理时用文件包含的正文件内容替换到实际程序中。

(1) 格式

文件包含伪指令的格式如下：

```
#include <头文件名.h>;标准头文件
#include"头文件名.h";自定义头文件
#include 宏标识符
```

(2) 说明

文件包含伪指令的说明如下。

① 常在头文件名后用.h 作为扩展名,可带或不带路径。

② 头文件可分为标准头文件和自定义头文件。

③ 尖括号内的头文件为标准头文件,由开发环境或系统提供。

④ 双引号内的头文件为用户自定义头文件。搜索时,首先在当前目录中搜索;其次按环境变量 include 指定的目录顺序搜索。

⑤ 搜索到头文件后,就将该伪指令直接用头文件内容替换。

(3) 举例

【例 3.1】 标准头文件定义。

```
#include <string.h>
#include <stdio.h>
```

string.h 和 stdio.h 是标准头文件,按环境变量 include 指定的目录顺序搜索 string.h 和 stdio.h。

【例 3.2】 用户自定义头文件定义。

```
#include "s3c2410-adc.h"
```

s3c2410-adc.h 头文件是用户自定义的有关三星 s3c2410 的 ARM 处理器的 A/D 转换器的各寄存器。

2. 宏定义伪指令

宏定义伪指令分为:简单宏、参数宏、条件宏、预定义宏及宏释放。

(1) 简单宏

格式如下:

```
#define 宏标识符　宏体
```

其中:

① 宏体由单词序列组成。宏体超长时,允许使用续行符"\"进行续行,续行符和其后的换行符 "\n"都不会进入宏体。

② 在定义宏时,应尽量避免使用 C 语言的关键字和预处理器的预定义宏,以免引起灾难性的后果。

③ 在源文件中,用预处理器伪指令定义过宏标识符之后,就可用宏标识编写程序。当源文件被预处理器处理时,每遇到该宏标识符,预处理器便将宏展为宏体。

(2) 参数宏

格式如下:

```
#define 宏标识符(形式参数表)　宏体
```

其中:

① 形式参数表为逗号分割的形式参数。

② 宏体由单词序列组成。宏体超长时，允许使用续行符“\”进行续行，续行符和其后的换行符“\n”都不会进入宏体。

③ 使用参数宏时，形式参数表应换为同样个数的实参数表，这一点类似于函数的调用。参数宏与函数的区别在于参数宏的形参数表中没有类型说明符，且在时空上的开销比函数要小。

④ 预处理器在处理参数宏时使用两遍宏展开。第 1 遍展开宏体；第 2 遍对展开后的宏体用实参数替换形式参数。

【例 3.3】 在 Linux 系统的/include/asm-arm/arch-s3c2410/S3C2410.h 头文件中定义了各 Nand Flash 控制寄存器，其源代码如下：

```
#define   bNAND_CTL(Nb)    __REG(0x4e000000+(Nb))
#define   NFCONF           bNAND_CTL(0x00)
#define   NFCMD            bNAND_CTL(0x04)
#define   NFADDR           bNAND_CTL(0x08)
#define   NFDATA           bNAND_CTL(0x0c)
#define   NFSTAT           bNAND_CTL(0x10)
#define   NFECC            bNAND_CTL(0x14)
```

【例 3.4】 在 Linux 下 ARM S3C2410X 芯片的 A/D 转换的驱动程序的头文件 s3c2410-adc.h 中定义了下面 3 个宏。

```
#define  ADC_WRITE(ch, prescale)  ((ch)<<16|(prescale))
                                                    /*ADC通道号与预标值合成一个字*/
#define  ADC_WRITE_GETCH(data)   (((data)>>16)&0x7)     /*获得ADC通道号*/
#define  ADC_WRITE_GETPRE(data)  ((data)&0xff)          /*获得ADC的预定标值*/
```

【例 3.5】 在 Linux 下 ARM S3C2410X 芯片的 A/D 转换的驱动程序实现代码 s3c2410-adc.c 中的系统资源和宏定义。

```
#define   DEVICE_NAME   "s3c2410-adc"       /*定义ADC设备的名字*/
#define   ADCRAW_MINOR  1
static   int  adcMajor=0;                   /*定义ADC设备的主设备号*/
typedef  struct {
    struct semaphore lock;
          /*内核信号量,当多个用户程序同时访问一个ADC控制器时,用lock进行同步*/
    wait_queue_head_t wait;                 /*等待队列*/
    int channel;                            /*ADC通道号*/
    int prescale;                           /*预定标值*/
}ADC_DEV;
static  ADC_DEV  adcdev;
#define  START_ADC_AIN(ch, prescale)\
do{  ADCCON=PRESCALE_EN|PRSCVL(prescale)|ADC_INPUT((ch));\
ADCCON|=ADC_START;\
}while(0)                  /*设置S3C2410X的ADC的通道为ch、预定标值为prescale*/
```

其中：

① PRESCALE_EN 宏对应 ARM S3C2410X 芯片的 A/D 转换控制寄存器的第 14

位 PRSCEMN，即 A/D 转换器预标器使能。

② PRSCVL 宏对应 ARM S3C2410X 芯片的 A/D 转换控制寄存器的第 6 位，设置预定标值。

③ ADC_INPUT 宏对应 ARM S3C2410X 芯片的 A/D 转换控制寄存器的第 3～5 位，选择通道号。

④ ADCCON|＝ADC_START：ADCCON 0 置为 1，准备采集数据。

(3) 条件宏

先测试是否定义过某宏标识符，然后决定如何处理。定义条件宏有两种格式。

格式 1：

```
#ifdef  宏标识符
    #undef    宏标识符
    #define   宏标识符    宏体
#else
    #define   宏标识符    宏体
#endif
```

格式 2：

```
#ifndef       宏标识符
    #define   宏标识符    宏体
#else
    #undef    宏标识符
    #define   宏标识符    宏体
#endif
```

其中：

① 格式 1 是测试存在，格式 2 是测试不存在。

② else 可有，也可没有。

(4) 宏释放

用于释放原先定义的宏标识符。经释放后的宏标识符可再次用于定义其他宏体。宏释放的格式如下：

```
#undef   宏标识符
```

【例 3.6】 定义与释放宏。

```
#define  SIZE  512
    ⋮
buf=SIZE*blks                  /*宏扩展为 buf=512*blks*/
    ⋮
undef SIZE
#define SIZE   128
    ⋮
buf=SIZE*blks                  /*宏扩展为 buf=128*blks*/
```

3. 条件编译伪指令

条件编译伪指令是写给编译器的，指示编译器在满足某一条件时仅编译源文件中与之相应的部分。预处理器对它的作用仅是扫描其中的宏并进行宏扩展，其他内容不动，留给编译器对它进行处理。

格式如下：

```
#if(条件表达式 1)
    ⋮
#elif(条件表达式 2)
    ⋮
#elif(条件表达式 3)
    ⋮
#elif(条件表达式 n)
    ⋮
#else
    ⋮
#endif
```

其中：

① 条件表达式允许使用宏标识符。

② 编译时，编译器仅对 # if()... # endif 之间满足某一条件表达式的源文件部分进行编译。

【例 3.7】

```
#if  _B0SIZE==B0SIZE_BYTE
    typedef unsigned char PB0SIZE;
#elif  _B0SIZE==B0SIZE_SHORT
    typedef unsigned short PB0SIZE;
#elif  _B0SIZE==B0SIZE_WORD
    typedef unsigned long PB0SIZE;
#endif
```

3.2 嵌入式 C 语言的基本数据类型

3.2.1 数据类型与表达式

嵌入式程序设计中 C 语言中的数据类型有基本类型和非基本类型，如图 3.1 所示。

1. 基本数据类型

基本数据类型：字符型(char)、整型(int)、单精度实型(float)、双精度实型(double)等。

无值型(void)有两种用途：第一是函数无返回值；第二是产生同一类型的指针。

(1) 类型修饰符

除了无值类型外，基本数据类型可以带有各种修饰前缀。修饰符用于明确基本数据

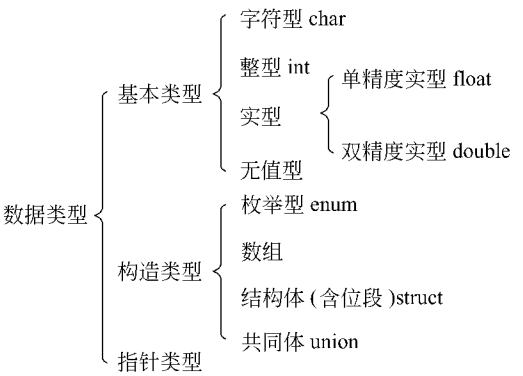

图 3.1　C 数据类型

类型的含义，以准确地适应不同情况下的要求。类型修饰符种类有：signed、unsigned、long 和 short。它们在内存中都是以二进制的形式存放的。每种类型的整数占有一定大小的地址空间，因此它们所能表示的数值范围也有所限制。

需要注意的是，不同的计算机体系结构中这些类型所占比特数有可能不同，如 ARM 32 位机中 C 语言常用的基本类型与修饰符的组合如表 3.1 所示。

表 3.1　C 语言常用的基本类型与修饰符的组合

类　　型	二进制位长度	值　　域
char	8	－128～127
unsigned char	8	0～255
signed char	8	－128～127
unsigned int	32	0～429496729
[signed] int	16	－2147483648～2147483647
[signed] short int	16	－32768～32767
long [int]	32	－2147483648～2147483647
unsigned long [int]	32	0～4294967295
float	32	3.4e－38～3.4e＋38(绝对值)
double	64	1.7e－308～1.7e＋308(绝对值)

(2) 访问修饰符

C 语言有两个用于控制访问和修改变量方式的修饰符，分别是常量(const)和易变量(volatile)。带 const 修饰符定义出的常量在程序运行过程中始终保持不变。例如：

```
const int num;
```

将产生整型常量 num，其值不能被程序所修改，但可以在其他类型的表达式中使用。const 型量可以在其初始化时直接被赋值，或通过某些硬件的方法赋值，例如：

```
const int num=100;
```

volatile 修饰符用于提醒编译程序，该变量的值可以不通过程序中明确定义的方法来改变。例如一个全程变量用于存储系统的实时时钟值，在这种情况下，变量的内容在程序中没有明确的赋值语句对它赋值时，也会发生改变。

const 和 volatile 可以同时使用。例如，假设 0x30 是一个只随外部条件而变化的口地址值，那么就恰好需要用下述说明来避免偶然因素所产生的副作用的影响。

```
const volatile unsigned char * port=0x30;
```

2. 构造数据类型

数组是一组连续、有序地存放在一起的具有相同类型的数据。

结构体是将不同类型的数据按一定顺序存放在一起的数据结构。

共用体是将不同类型的数据都存放在同一起始地址的内存单元中，共用一段内存以节省内存单元。

枚举是只有几种可能的值，将其一一列举出来。实际是用符号来表示若干个可取的整型值，它是整型的一个子集。

3. 指针类型

指针是 C 语言中一个重要概念。正确而灵活地运用它，可以有效地表示复杂的数据结构；能动态分配内存；能方便地使用字符串；有效而方便地使用数组；在调用函数时能得到多于一个的值；能直接处理内存地址等。

指针类型迥异于前述各种数据类型，不管是简单类型的数据，还是构造类型数据，均是代表数据的，而指针类型是代表地址的。

3.2.2 常量

1. 数值常量

(1) 整型常量

整型常量也称为整型常数或整数。

C 整型常量按进制分可分为十进制整数、八进制整数和十六进制整数。

① 十进制整数。十进制整数以正负号开头，后跟 0～9 的若干位数字。如 123，－456，0 等。

② 八进制整数。八进制整数是以正负号开头，第一位数字一定是 0，后面跟 0～7 的数字。如八进制数 0123，相当于十进制数 83；八进制数－012，相当于十进制数－10。

③ 十六进制整数。十六进制整数是以正负号开头，前两位为 0x，后面跟 0～9 和 a～f 的数字。其中 a 代表 10，b 代表 11，…，f 代表 16。

下面示例的几句代码是从 Linux 内核源程序中摘录出来的(/arch/arm//mach-s3c2410)，读者在这里先暂时不用了解这些代码的具体含义，而只需了解这些常量的表示方法。

```
unsigned loog s3c_irqwake_eintallow=0x0000fff0L;        /* 此代码在(irq.c) */
```

```
if (pinstate==0x02) {...}                    /* 此代码在(pm.c)中 */
config &=0xff;                               /* 此代码在(gpio.c)中 */
```

可以看出，第一句代码是使用常量 0x0000fff0L 对变量 s3c_irqwake_eintallow 进行赋值；第二句是比较变量 pinstate 的值和 0x02 是否相等；第三句则是对 config 进行特定的运算。从第一句还可以看出，整型常量在结尾加上 L 或 l 代表长整型，若为 U 或 u 代表无符号整型。

(2) 实型常量

实型常量有单精度实型常量和双精度实型常量。可用小数形式或指数形式表示。

(3) 字符常量

字符常量是指用单引号括起来的一个字符，如'a'、'M'和'?'是字符常量，而"a"、"M"和"?"等是字符串常量。

注意：字符常量只能用单引号括起来，而不能用双引号或其他括号；字符常量只能是单个字符，而字符串可以是字符集中任意字符。

数字若被定义为字符型就不能参与数值运算，如'5'和 5 是不同的，'5'是字符常量，不能直接参与运算，而只能以其 ASCII 码值(0x35)来参与运算。

除此之外，C 语言中还存在一种特殊的字符常量——转义字符，如表 3.2 所示。

表 3.2　控制字符表示法

字 符 形 式	ASCII 码(十六进制)	功　　能
\n	0A	换行
\t	09	横向跳格(即跳到下一个输出区)
\v	0B	竖向跳格
\b	08	退格
\r	0D	回车
\a	07	响铃
\\	5C	反斜杠字符“\”
\'	27	单引号
\"	22	双引号
\ddd	Ddd(八进制)	1～3 位八进制数所代表的字符
\xhh	Hh	1～2 位十六进制数所代表的字符

2. 字符串常量

字符串常量简称字符串，是用一对双引号括起来的字符序列，例如"China"就是一个字符串常量。

3. 符号常量

(1) 不带参数的宏定义

宏定义命令 # define 的一般形式是：

```
#define 宏名  字符串
```

终止宏名作用域命令 #undef 的一般形式是：

```
#undef 宏名
```

【例 3.8】

```
#define  PI  3.14159          /*定义 PI 为常量,其值是 3.14159*/
main()
  { ...
  }
#undef  PI                    /*终止宏名 PI 的作用域*/
f1()
```

(2) 带参数的宏定义

它不是进行简单的字符串替换，还要进行参数替换。其定义的一般形式为：

```
#define 宏名(参数表) 字符串
```

其中字符串中包括参数表中所指定的参数。在使用时，要将程序中宏名后的实际参数代入字符串中参数的位置。例如：

```
#define S(a, b)  a*b
    ⋮
area=S(3, 2);
```

经编译预处理，该语句被展开成：

```
area=3*2;
```

说明：

① 宏名和参数表左括号之间不能有空格；否则按不带参宏替换了。

② 字符串中应注意括号的使用，以保证运算次序。如上例改成：

```
area=S(1+2, 2);
```

经展开后变成：

```
area=1+2*2;
```

这就不合要求了。此时，可改写成：

```
#define S(a, b) (a)*(b)
    ⋮
area=S(1+2, 2)
```

经展开后变成：

```
area=(1+2)*(2);
```

就不会出现错误了。

3.2.3　变量

1. 变量的定义

变量定义的一般形式如下：

```
[存储类型]  类型说明符  [修饰符]  标识符  [=初值][,标识符[=初值]]…;
```

变量的定义由 5 部分组成，方括号中的可有可无，视变量定义的具体情况而定，下面对变量定的 5 部分作如下说明。

(1) 类型说明符

① 对于数字与字符，其常用的类型主要有 8 种：char、unsigned char、int、unsigned、long、unsigned long、float、double。

② void 类型(抽象型)，在具体化时可用类型强制来指定类型说明符中的任意一类。

③ 对于通过 typedef 定义的类型别名，为了增加程序的可读性和移植程序时的方便，C 语言允许用户为 C 语言固有的类型用 typedef 起别名。

格式如下：

```
typedef   C语言固有的简单类型或复合类型别名标识符;
```

用别名代替原来的类型，在说明中用作类型说明符。别名一般用大写字符，例如：

```
typedef long BIG
BIG  x=80000;
```

(2) 标识符

① 变量名可以是 C 语言中允许的合法标识符，用户定义时应遵循“见名知意”的原则，以利于程序的维护。

② 每一个变量都必须进行类型说明，这样就可以保证程序中变量的正确使用。未经类型说明的变量在编译时将被指出是错误的，也就是变量要先定义，后使用。

③ 当一个变量被指定为某一确定类型时，将为它分配若干相应字节的内存空间。如在 32 位体系的 ARM 系统中，char 型为 1 字节，int 型为 4 字节，float 为 4 字节，double 为 8 字节。当然，不同的体系结构的系统可能稍有差异。

④ 变量可以在程序内的 3 个地方定义：在函数内部，在函数的参数定义中或在所有的函数外部。由此定义的变量分别称为局部变量、形式参数和全局变量。在不同地方定义的变量，其作用域范围不同。在同一层次定义的变量不能与数组、指针、函数和其他变量同名。

⑤ 变量是用来存放数据的，由于数据有不同的类型，因此要定义相应类型的变量去存放它。这些数据称为相应变量的值。

(3) 存储类型

存储类型指定被说明对象所在内存区域的属性。

内存中供用户使用的存储空间分为代码区与数据区两个部分。变量存储在数据区，数据区又可分为静态存储区与动态存储区。