TCR 嵌入式培训--C 语言基础复习

在学习 STM32 的视频前，我应该先让你们复习一下知识的（暴风哭泣），我的错。现在复习还来得及！！！本文还是比较通俗易懂的！大家加油鸭！

这节主要讲解 C 语言基础知识。这里主要是简单复习几个 C 语言的基础知识点，引导那些 C 语言基础知识不是很扎实的同学能够快速开发 STM32 程序。同时希望这些同学能够多复习 C 语言基础知识，C 语言毕竟是单片机开发中的必备基础知识。本人 C 语言挂科，再次暴风哭泣。

1. 位操作

C 语言位操作，简而言之，就是对基本类型变量可以在位级别进行操作。C 语言支持如表 1 所列的 6 种位操作。下面着重讲解位操作在单片机开发中的一些实用技巧。

运算符	含义	运算符	含义
&	按位与	~	取反
|	按位或	<<	左移
^	按位异或	>>	右移

表 1

① 不改变其他位的值的状况下，对某几个位进行设值。

这个场景在单片机开发中经常使用，方法就是先对需要设值的位用“&”操作符进行清零操作，然后用“|”操作符设值。比如要改变 GPIOA->BSRRL 的状态，可以先对寄存器的值进行“&”清零操作：

```
GPIOA->BSRRL &= 0XFF0F;

//将第 4-7 位清 0
```

然后再与需要设置的值进行“|”运算：

```
GPIOA->BSRRL |= 0X0040;

//设置相应位的值，不改变其他位的值
```

② 移位操作提高代码的可读性。

移位操作在单片机开发中也非常重要，看下面一行代码：

```
GPIOx->ODR = (((uint32_t)0x01) << pinpos);
```

这个操作就是将 ODR 寄存器的第 pinpos 位设置为 1，为什么要通过左移而不是直接设置一个固定的值呢？其实，这是为了提高代码的可读性以及可重用性。这行代码可以很直观地知道是将第 pinpos 位设置为 1。如果写成：

```
GPIOx->ODR = 0x0030;
```

这样的代码不好看也不好重用了。

③ ~取反操作使用技巧

SR 寄存器的每一位都代表一个状态，某个时刻我们希望设置某一位的值为 0，同时其

他位都保留 1，简单的方法是直接给寄存器设置一个值：

```
TIMx->SR = 0xFFF7;
```

这样的做法可以设置第三位为 0，但是这样的方法同样不好看，并且可读性很差。看看库函数代码中怎样使用的：

```
TIMx->SR = (uint16_t)~TIM_FLAG;
```

而 TIM_FLAG 是通过宏定义定义的值：

```
#define TIM_FLAG_Update          ((uint16_t)0x0001)

#define TIM_FLAG_CC1             ((uint16_t)0x0002)
```

看这个应该很容易明白，可以直接从宏定义中看出 TIM_FLAG_Update 就是设置的第 0 位了，可读性非常强。

2. define 宏定义

define 是 C 语言中的预处理命令，用于宏定义，可以提高源代码的可读性，为编程提供方便。常见的格式：

```
#define    标识符          字符串
```

其中，“标识符”为定义的宏名，“字符串”可以是常数、表达式、格式串等。例如：

```
#define    PLL_M           8

//定义标识符 PLL_M 的值为 8
```

至于 define 宏定义的其他一些知识，比如宏定义带参数这里就不多讲解。

3. ifdef 条件编译

单片机程序开发过程中，经常会遇到一种情况：当满足某条件时对一组语句进行编译，而当条件不满足时则编译另一组语句。条件编译命令最常见的形式为：

```
#ifdef 标识符

程序段 1

#else

程序段 2

#endif
```

它的作用是：当标识符已经被定义过（一般是用#define 命令定义），则对程序段 1 进行编译，否则编译程序段 2。其中#else 部分也可以没有，即：

```
#ifdef

程序段 1

#endif
```

这个条件编译在 MDK 里面是用的很多的，在 stm32f4xx.h 头文件中经常会看到这样的语句：

```
#if   defined(STM32F40_41xxx)

//STM32F40x 系列和 STM32F41x 系列芯片需要的一些变量定义
```

#end

而 STM32F40_41xxx 则是我们通过#define 来定义的。条件编译也是 C 语言的基础知识，这里也就点到为止吧。

4. extern 变量申明

C 语言中 extern 可以置于变量或者函数前，以表示变量或者函数的定义在别的文件中，提示编译器遇到此变量或函数时在其他模块中寻找其定义。注意，对于 extern 申明变量可以多次，但定义只有一次。在我们的代码中会看到这样的语句：

```
extern u16 USART_RX_STA;
```

这个语句是申明 USART_RX_STA 变量在其他文件中已经定义了，这里要使用到。所以，你肯定可以找到在某个地方有变量定义的语句：

```
u16 USART_RX_STA;
```

的出现。下面通过一个例子说明一下使用方法。

在 main.c 定义的全局变量 id，其初始化都是在 main.c 里面进行的。main.c 文件：

```
u8 id;

main()

{

    id = 1;

    printf("d%",id);//id = 1
```

```
    test();

    printf("d%",id);//id = 2

}
```

但是我们希望在 test.c 的 test(void)函数中使用变量 id，这个时候就需要在 main.c 里面去申明变量 id 是外部定义的了，因为如果不申明，变量 id 的作用域到不了 test.c 文件中。看下面 main.c 中的代码：

```
extern u8 id;

void test(void)

{

    id = 2;

}
```


在 main.c 中申明变量 id 在外部定义，然后在 test.c 中就可以使用变量 id 了。extern 申明函数在外部定义的应用这里就不多讲解了。

5. typedef 类型别名

typedef 用于为现有类型创建一个新的名字，或称为类型别名，用来简化变量的定义。typedef 在 MDK 中用的最多的就是定义结构体的类型别名和枚举类型了。

```
struct  _GPIO

{
```

```
    __IO uint32_t MODER;
    __IO uint32_t OTYPER;
    ...
};
```

定义了一个结构体 GPIO，这样我们定义变量的方式为：

```
struct  _GPIO  GPIOA//定义结构体变量 GPIOA
```

但是这样很繁琐，MDK 中有很多这样的结构体变量需要定义。这里可以为结构体定义一个别名 GPIO_TypeDef，这样就可以在其他地方通过别名 GPIO_TypeDef 来定义结构体变量了。方法如下：

```
typedef  struct
{
    __IO uint32_t MODER;
    __IO uint32_t OTYPER;
    ...
}GPIO_TypeDef;
```

typedef 作为结构体定义一个别名 GPIO_TypeDef，这样可以通过 GPIO_TypeDef 来定义结构体变量：

```
GPIO_TypeDef  _GPIOA,_GPIOB;
```

这里的 GPIO_TypeDef 就与 struct　_GPIO 是等同的作用了，这样是不是方便很多？

6. 结构体

声明结构体类型：

struct 结构体名{

成员列表

}变量名列表;

例如：

struct U_TYPE{

Int BaudRate;

Int WordLength;

}usart1,usart2;

在结构体声明的时候可以定义变量，也可以声明之后定义，方法是：

struct　结构体名　结构体变量列表;

例如：

struct　U_TYPE　usart1,usart2;

结构体成员变量的引用方法是：

结构体变量名.成员名

比如要引用 usart1 的成员 BaudRate，方法是："usart1.BaudRate"。结构体指针变量定义也是一样的，跟其他变量没有啥区别。例如：

```
struct  U_TYPE  *  usart3;//定义结构体指针变量 usart3
```

结构体指针成员变量引用是通过"->"符号实现，比如要访问 usart3 结构体指针指向的结构体的成员变量 BaudRate，方法是：

```
usart3->BaudRate;
```

上面讲解了结构体和结构体指针的一些知识，其他的这里就不多讲解了。讲到这里，有人会问，结构体到底有什么作用呢？下面通过一个实例来回答。

在单片机程序开发过程中，经常会遇到要初始化一个外设比如串口，它的初始化状态是由几个属性来决定的，比如串口号、波特率、极性以及模式等。对于这种情况，在没有学习结构体的时候，我们的一般方法是：

```
void USART_Init(u8 USARTx,u32 BaudRate,u8 Parity,u8 Mode);
```

这种方式是有效的，并且在一定场合是可取的。但是试想，如果有一天，我们希望往这个函数里再传入一个参数。于是我们的定义被改为：

```
void USART_Init(u8 USARTx,u32 BaudRate,u8 Parity,u8 Mode,u8 WordLength);
```

如果这个函数的入口参数随着开发不断增多，那么我们是不是就要不断地修改函数的定义呢？使用结构体就可以解决这个问题了。我们可以在不改变入口参数的情况下，只需要改变结构体的成员变量，就可以达到上面改变入口参数的目的。

结构体就是将多个变量组合为一个有机的整体。上面函数中 usartx、BaudRate、

wordlength、parity、mode 这些参数，对于串口而言，是一个有机整体，都是用来设置串口参数的，所以可以将它们通过定义一个结构体来组合。MDK 中是这样定义的：

```
typedef struct
{
   uint32_t USART_BaudRate;
   uint16_t USART_WordLength;
   uint16_t USART_StopBits;
   uint16_t USART_Parity;
   uint16_t USART_Mode
   uint16_t USART_HardwareFlowControl;
} USART_InitTypeDef;
```


于是，在初始化串口的时候入口参数就可以是 USART_InitTypeDef 类型的变量或者指针变量了，MDK 中是这样做的：

```
void USART_Init(USART_TypeDef * USARTx,
                USART_InitTypeDef * USART_InitStruct);
```

这样，任何时候，我们只需要修改结构体成员变量，往结构体中间加入新的成员变量，而不需要修改函数定义就可以达到修改入口参数同样的目的了。

C 语言的基础差不多复习完了，很多细节大家还需要自己发现鸭~

本文部分文字叙述引用自

《精通 STM32F4（库函数版）》P71-P76

MAR/27/2020

Author:Chen