(79 条消息) 【C 语言】模块化编程-通俗易懂_Z 小旋-CSDN 博客_模块化编程

1什么是模块化

模块化编程就是把我们的一整个项目，分成很多模块(比如一个学生成绩查询可以分为，登陆，查询，修改保存，退出等模块)

而一个程序工程包含多个源文件（.c 文件和 .h 文件），每个 .c 文件可以被称为一个模块，每一个模块都有其各自的功能，而每一个.h文件则是声明该模块，相当于功能说明书　模块化编程在嵌入式中为必须要掌握的技能

2为啥要用模块化

有的同学会想，我一个main.c也写得津津有道的，为什么偏要分开呢。

在我们实际应用中，当你的代码长度长起来了以后就会发现，想自己以前的代码里面找到之前定义的模块很麻烦，因为代码太多太繁杂了，你很难有一个清晰的分类，这就导致了代码的臃肿性，并且别人也很难看懂你的代码。

并且在实际项目开发的时候，一个复杂的项目意味着你需要和别人组成小组一起进行开发，这时候每个人负责一部分功能的开发，而你所负责的模块，你需要将你负责的模块功能写好，封装好，之后形成一个.c与.h 然后交付给项目组长，组长则负责整体框架(main)的编写与各个模块的组合调试，最后各个模块的组合，形成了整个工程。

这时候就可以彰显模块化的作用了，它使得整个项目分工明确，条理清晰，易于阅读，便于移植，等优点

模块化具体原理：

我们在写C语言代码的时候，首先做的就是**引入头文件**

```
1  #include<stdio.h>
2  int main()
```

在相对应的头文件引入之后，就可以使用相对应头文件里的函数，

比如　#include<stdio.h>

之后我们就可以使用printf scanf 语句进行数据的打印与获取，而printf和scanf语句的定义则是在stdio.h中，用户只需要负责调用即可

模块化编程的核心思想也正是如此：　将系统的各个功能进行封装，变成一个个独立的模块，其他人只需要使用你所提供的函数和变量等，就可以完成相对应的功能

模块化的本质也就是，新建一个.c和.h文件，

.c文件里面存放着你所构建的函数，功能，等等，而当你想让他可以被其他文件使用时，这时候便需要在对应的.H之中声明，

在外部调用时，只需要#include包含相对应的.h 即可

3模块化基本代码实现：

我们以最简单的LED为例 ，将其分为一个模块

LED.h

```
1  #ifndef LED.h
2
3  #define LED.h
4
5  extern void LED_Open();   //开启LED灯
6
7  extern void LED_Close();  //关闭LED灯
8
9  #endif
```

LED.c

```
void LED_Open()

{
  //亮灯代码

}
void LED_Close()

{
  //关灯代码

}
```

main.c 主函数

```
#include "LED.h"

int main(void)
{
    LED_Open();  //开启LED灯

    while(1);
}
```

这样子你的LED部分的代码就会独立起来，需要使用时直接调用函数即可，修改也会变得十分简便

模块化的核心也就是各个模块独立封装，多个.c和.h 使得整个工程变得易于阅读，逻辑清晰

我们分布讲解

首先

#ifndef XXX 表示如果没有定义 xxx 则执行后面的语句 如果已经定义则不执行，

#define xxx 定义一个预处理宏定义，

#endif 表示条件命令的结束

我们这里#ifndef LED.h #define LED.h 表示如果没有定义LED.H这个头文件，则定义LED.h 并且后面的语句都有效，直到#endif 结束命令为止

同时声明了开LED灯和关LED灯两个函数

具体格式为：

```

#ifndef _XXX_h_

#define _XXX_h_

#endif

```

.c文件中：

```
#include "XXX.h"
```

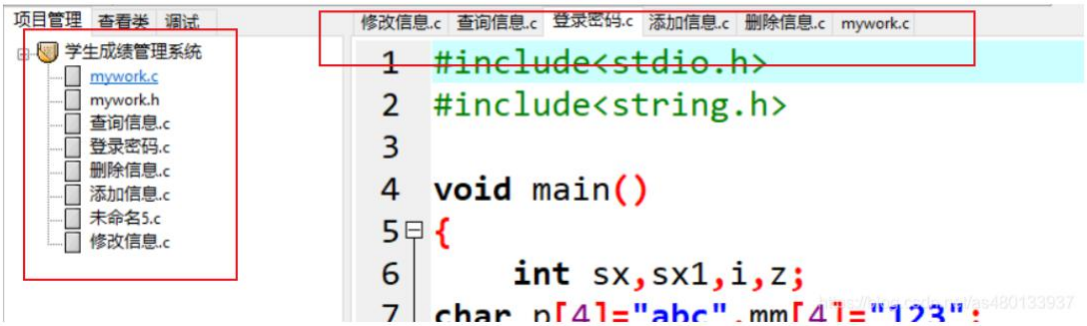
.C文件

之后LED.c文件则是你所构建的函数，完成函数功能的编写，和变量的定义

最后**在主函数或者其他函数中 #include LED.h 包含头文件，即可调用相对应声明的函数和变量**

这便是一个模块的构建，而构建多个模块实现其各自功能，并且在主函数中分别调用，这便是模块化编程

比如我想要建立一个学生成绩管理系统，就可以分成几个模块，**分别建立相对应的.c文件和.h文件，最后在主函数中调用相对应功能即可**



c语言中条件编译相关的预编译指令

#define	*定义一个预处理宏*
#undef	*取消宏的定义*
#if	*编译预处理中的条件命令，相当于C语法中的if语句*
#ifdef	*判断某个宏是否被定义，若已定义，执行随后的语句*
#ifndef	*与#ifdef相反，判断某个宏是否未被定义*
#elif	*若#if, #ifdef, #ifndef或前面的#elif条件不满足，则执行#elif之后的语句，相当于C语法中的else-if*
#else	*与#if, #ifdef, #ifndef对应, 若这些条件不满足，则执行#else之后的语句，相当于C语法中的else*
#endif	*#if, #ifdef, #ifndef这些条件命令的结束标志.*
defined	*与#if, #elif配合使用，判断某个宏是否被定义*

具体可以阅读：【C语言】----宏定义，预处理宏

4模块化编程注意事项

头文件(XX.h)注意事项：

1.**函数默认是extern属性** 也就是我们声明函数的时候前面的extern可有可无

```
extern void LED_Open();
 void LED_Open();   //相同
```

2.“.h”文件中不可以定义变量 在.h中只能声明，不能定义

```
#ifndef LED.h
#define LED.h

extern a;  //声明变量a   正确
b=3;        //定义变量b   错误

#endif
```

3声明变量不会占用内存，定义变量才会

定义变量和声明变量的区别在于定义会产生分配内存的操作，这是汇编阶段的概念；声明则只是告诉包含该声明的模块在连接阶段从其他模块寻找外部函数和变量。

4 **不想让外界知道的信息，就不应该出现在头文件里，而想让外部调用的函数或变量，则一定要在头文件中声明**

5 头文件(.h)命名应与源文件(.c)一致，便于清晰的查看各个头文件

6 #include <stdio.h>,#include "myfile.h",双引号先在工程目录里寻找，再去系统目录里寻找。

.c文件(XX.c)注意事项：

1.**模块内不想被外部引用的函数和全局变量需在“.c”文件头冠以static关键字声明。** 这样这些函数和变量只会在当前.c文件中起到作用，一来可以避免函数名的重复；二来可以保护内部的实现数据，防止被破坏

```
static a = 3;
static void LED_Open();
```

2模块中想要被其他文件访问的变量，一定要是全局变量，并且在.h中声明

3 **要尽量减少全局变量的使用**，因为全局变量的生命周期是从程序的开始到程序的结束的，这意味着你在其他源文件中调用这个变量时，可能会产生同名变量，以及变量数值错误等问题

4函数的声明有无extern都行，变量的声明必须加上extern，否则编译器无法识别声明。