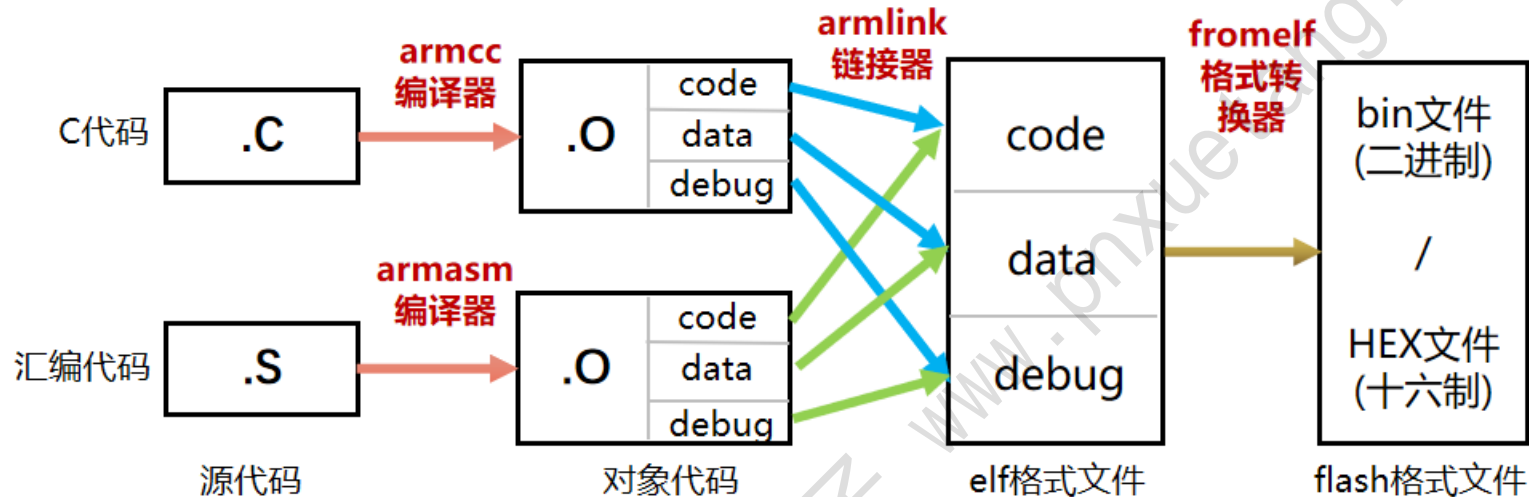


嵌入式C语言之- 多源文件的函数声明和定义

讲师：叶大鹏

助力你成为优秀的电子工程师！

编译过程简介



- (1) **编译**, MDK 软件使用的编译器是 armcc 和 armasm, 它们根据每个 c/c++ 和汇编源文件编译成对应的以 “.o” 为后缀名的对象文件 (Object Code, 也称目标文件), 其内容主要是从源文件编译得到的机器码, 包含了代码、数据以及调试使用的信息;
- (2) **链接**, 链接器 armlink 把各个.o 文件及库文件链接成一个映像文件 “.axf”, 它是elf格式文件;
- (3) **格式转换**, 一般来说 Windows 或 Linux 系统使用链接器直接生成可执行映像文件 elf 后, 内核根据该文件的信息加载后, 就可以运行程序了, 但在单片机平台上, 需要把该文件的内容加载到芯片上, 所以还需要对链接器生成的 elf 映像文件利用格式转换器 fromelf 转换成 “.bin” 或 “.hex” 文件, 交给下载器下载到芯片的 FLASH 或 ROM 中。

在同一个C文件中，函数的定义和声明

```
void AFunc(void)
{
    printf("This is AFunc.\n");
}
```

这段代码既完成了函数的声明，也实现了函数的定义

```
int main(void)
{
    AFunc();
    return 0;
}
```

在同一个C文件中，函数的定义和声明

```
int main(void)
{
    AFunc();
    return 0;
}
```

```
void AFunc(void)
{
    printf("This is AFunc.\n");
}
```

- 编译报错：

main.c(43): error: #159: declaration is incompatible with previous "AFunc" (declared at line 39)

```
compiling main.c...
```

```
main.c(40): warning: #223-D: function "AFunc" declared implicitly
```

```
    AFunc();
```

```
main.c(44): error: #159: declaration is incompatible with previous "AFunc" (declared at line 40)
```

```
void AFunc(void)
```

在同一个C文件中，函数的定义和声明

```
void AFunc(void)
{
    printf("This is AFunc.\n");
}
```

```
int main(void)
{
    AFunc();
    return 0;
}
```



```
int main(void)
{
    AFunc();
    return 0;
}
```

```
void AFunc(void)
{
    printf("This is AFunc.\n");
}
```



在同一个C文件中，函数的定义和声明

```
void AFunc(void);
```

这段代码完成了函数的声明

```
int main(void)
{
    AFunc();
    return 0;
}
```

```
void AFunc(void)
{
    printf("This is AFunc.\n");
}
```

在同一个C文件中，函数的定义和声明

```
void AFunc(void);
```

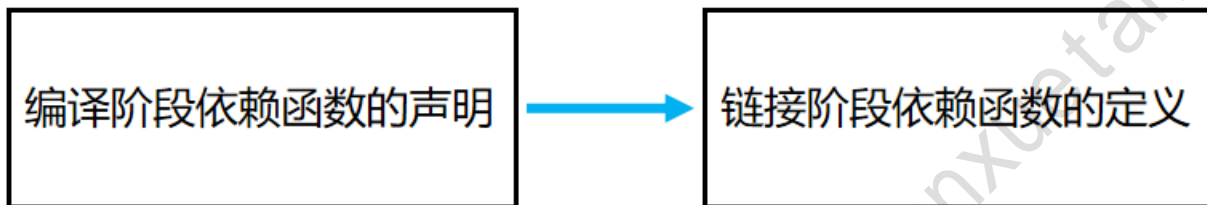
只有函数的声明?

```
int main(void)
{
    AFunc();
    return 0;
}
```

```
linking...
```

```
.\Objects\template.axf: Error: L6218E: Undefined symbol AFunc (referred from main.o).
```

函数的定义和声明



- 声明一个函数意味着向编译器描述函数名、返回值、参数个数和类型，但并不会为函数分配存储空间。
- 定义一个函数意味着在声明变量的同时还要有具体的实现，并且会为函数分配存储空间。

在同一个C文件中，函数的定义采用哪种方式？

```
void AFunc(void)
{
    printf("This is AFunc.\n");
}

int main(void)
{
    AFunc();
    return 0;
}
```

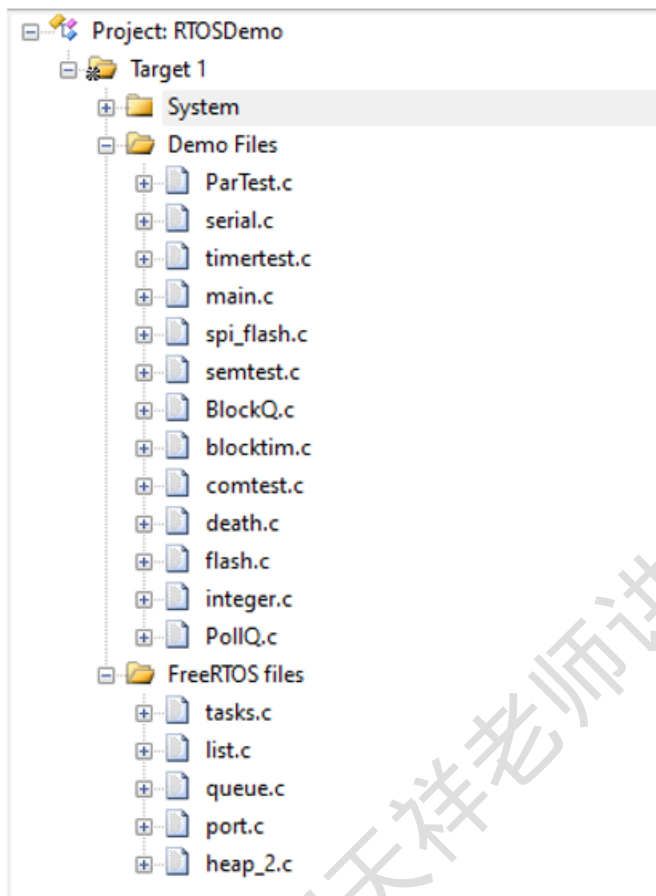
```
void AFunc(void);

int main(void)
{
    AFunc();
    return 0;
}

void AFunc(void)
{
    printf("This is AFunc.\n");
}
```

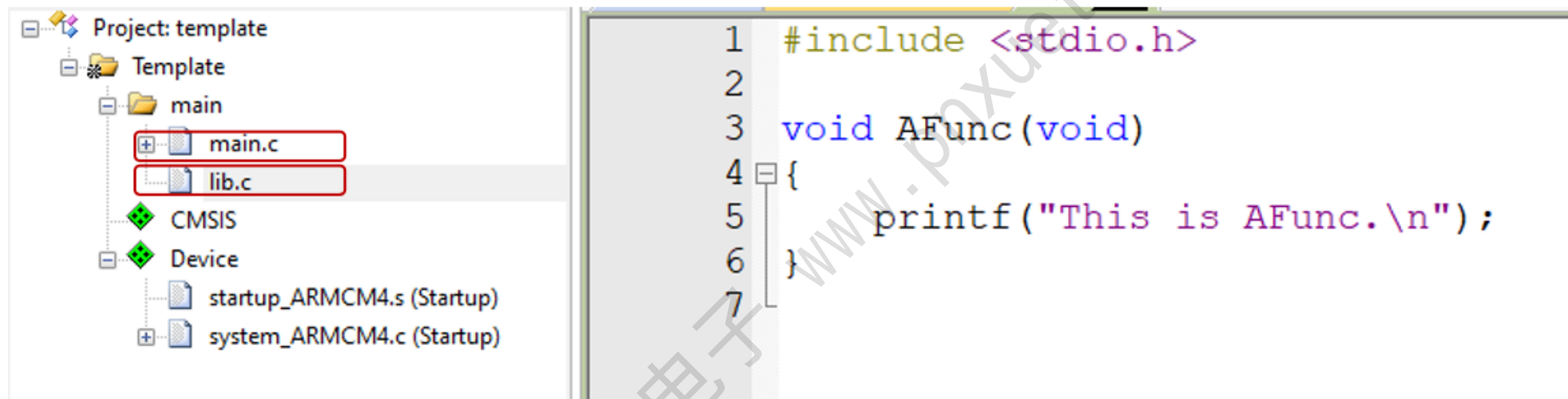
- 没有明确的标准，每个公司的规范都不尽相同，建议采用第一种方式，代码会更简洁，便于阅读。

为什么多源文件开发?

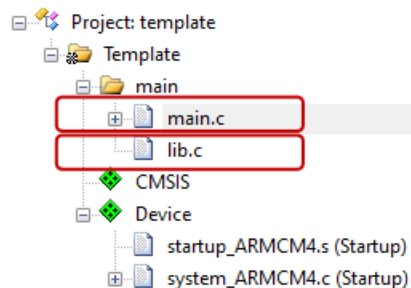


1. 便于团队合作，提升效率，同时开发一个源文件，会导致编写的代码冲突；
2. 代码结构清晰，每个源文件对应一个功能模块，便于维护和阅读。

在不同C文件中，函数的定义和声明



在不同C文件中，函数的定义和声明



```
18     __NOP();
19     return (ITM_ReceiveChar());
20 }
21 int ferror(FILE *f)
22 {
23     /* Your implementation of ferror */
24     return EOF;
25 }
26
27 void _ttywrch(int ch)
28 {
29     fputc(ch, &__stdout);
30 }
31
32 void _sys_exit(int return_code)
33 {
34     while (1);    /* endless loop */
35 }
36
37 int main(void)
38 {
39     AFunc();
40     return 0;
41 }
```

在不同C文件中，函数的定义和声明

- 能否编译通过？
- 能否正常运行？

在不同C文件中，函数的定义和声明

✓ 编译可以通过，但是有警告：

```
compiling main.c...
```

```
main.c(38): warning: #223-D: function "AFunc" declared implicitly  
    AFunc();
```

在不同C文件中，函数的定义和声明

✓ 运行正常：

Debug (printf) Viewer

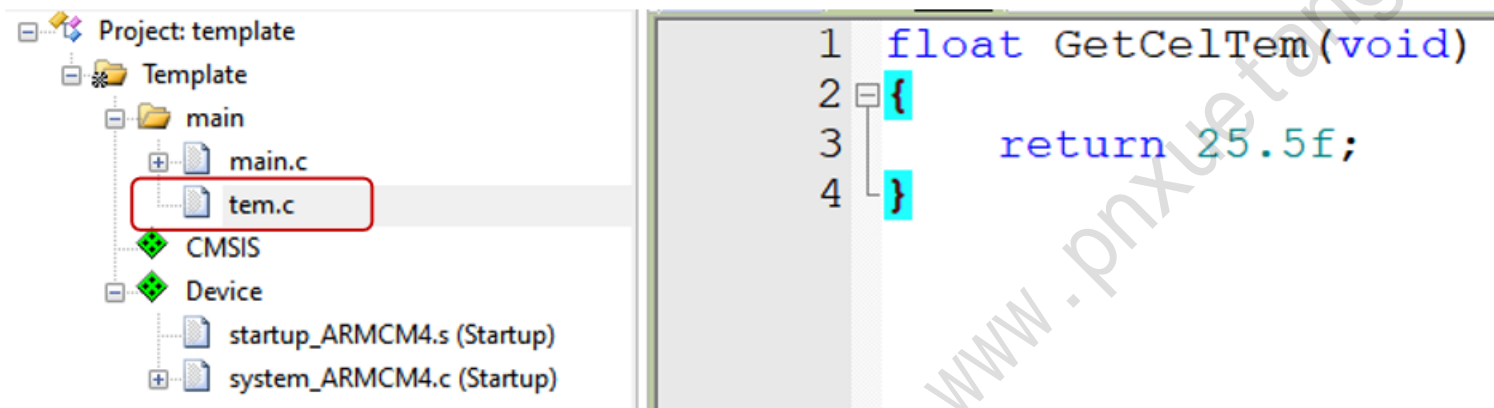
This is AFunc.

在不同C文件中，函数的定义和声明

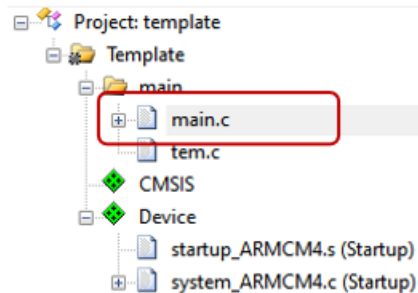
- 编译没有报错，是因为lib.c文件先于main.c文件被编译，已经完成了函数的声明和定义。

```
Rebuild started: Project: template
*** Using Compiler 'V5.06 update 6 (build 750)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Rebuild target 'Template'
assembling startup_ARMCM4.s...
compiling lib.c...
compiling system_ARMCM4.c...
compiling main.c...
main.c(39): warning: #223-D: function "AFunc" declared implicitly
    AFunc();
main.c: 1 warning, 0 errors
linking...
Program Size: Code=528 RO-data=992 RW-data=16 ZI-data=1024
After Build - User command #1: fromelf --text -a -c --output=all.dis Objects\Template.axf
".\Objects\template.axf" - 0 Error(s), 1 Warning(s).
```


在不同C文件中，函数的定义和声明



在不同C文件中，函数的定义和声明



```
17     __NOP();
18     return (ITM_ReceiveChar());
19 }
20 int ferror(FILE *f)
21 {
22     /* Your implementation of ferror */
23     return EOF;
24 }
25
26 void _ttywrch(int ch)
27 {
28     fputc(ch, &__stdout);
29 }
30
31 void _sys_exit(int return_code)
32 {
33     while (1);    /* endless loop */
34 }
35
36 int main(void)
37 {
38     float celTem;
39     celTem = GetCelTem();
40     printf("Temperature is %.1f cel degrees.\n", celTem);
41     return 0;
42 }
```

在不同C文件中，函数的定义和声明

- 能否编译通过？
- 能否正常运行？

在不同C文件中，函数的定义和声明

✓ 运行错误：

Debug (printf) Viewer

Temperature is 1103888384.0 cel degrees.

头文件

- 可以将函数的声明放到头文件（以.h结尾）中，这样在需要调用这个函数的C文件中使用 `#include` 来展开这个头文件，编译器在编译的时候就知道函数的声明了。

#include预处理指令

- #include是一个预处理指令，预处理这个动作发生在编译之前：



- #include的作用是，在预处理时，将文件中的全部文本内容全部复制粘贴到#include所在的位置；

```
main.c  tem.h
1
2 float GetCelTem(void);
3
```

```
main.c  tem.h
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "ARMCM4.h"
4 #include "tem.h"
```



```
main.c  tem.h
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "ARMCM4.h"
4 float GetCelTem(void);
```

应用案例

➤ 需求:

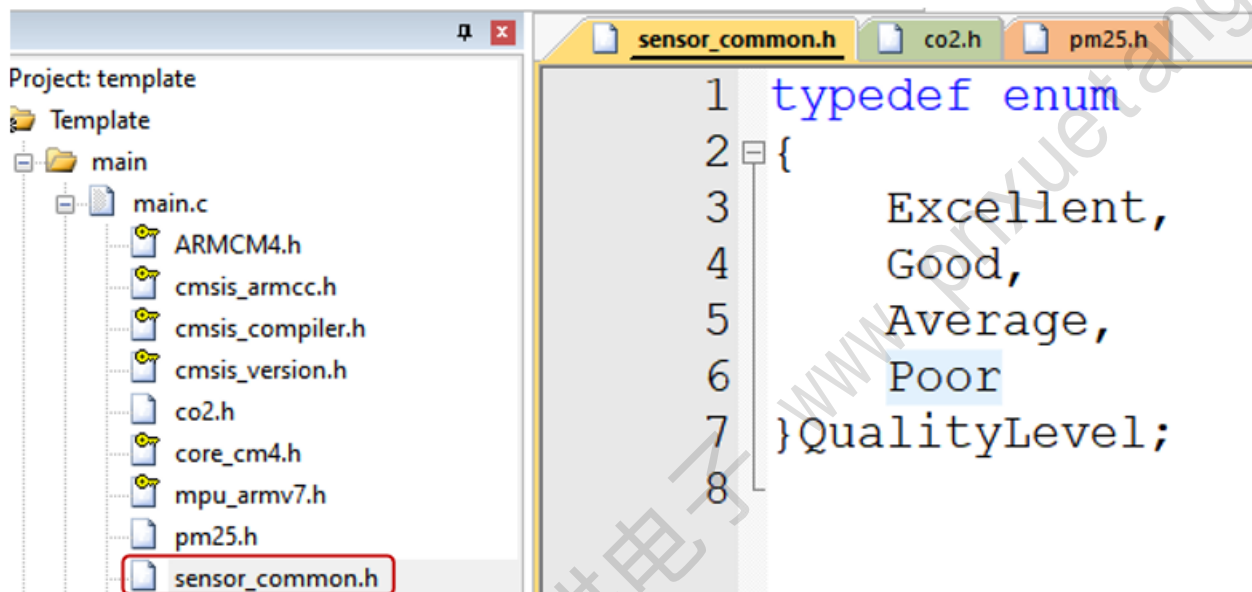
1. 根据CO2浓度的原始值, 计算并打印输出对应的标准等级, 公式为:

小于100	对应	Excellent
大于等于100、小于200	对应	Good
大于等于200、小于300	对应	Average
大于等于300	对应	Poor

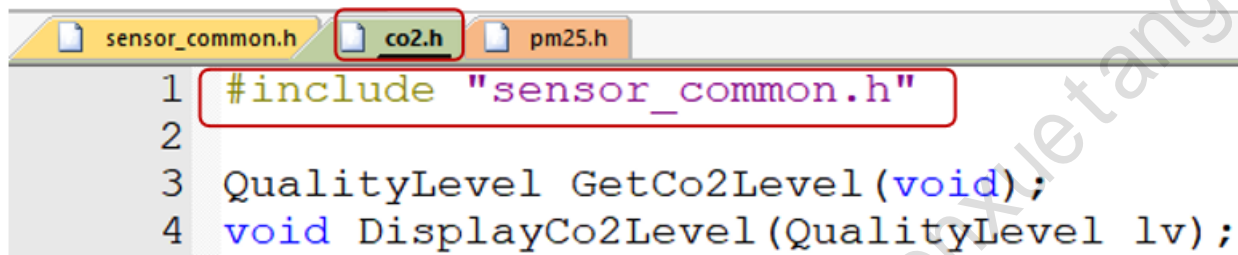
2. 根据PM2.5浓度的原始值, 计算并打印输出对应的标准等级, 公式为:

小于10	对应	Excellent
大于等于10、小于35	对应	Good
大于等于35、小于75	对应	Average
大于等于75	对应	Poor

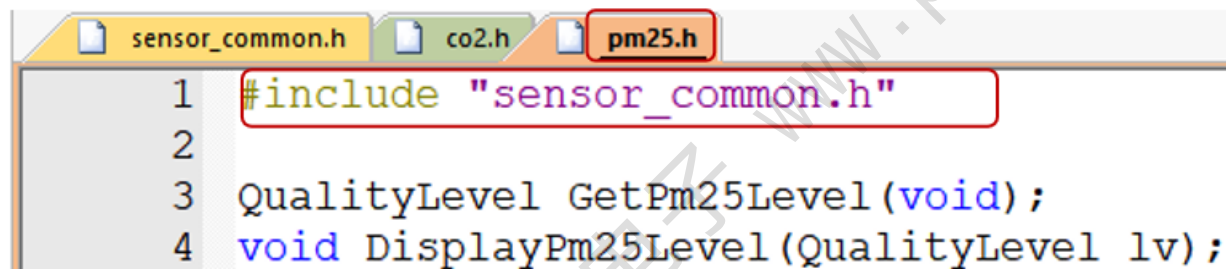
应用案例



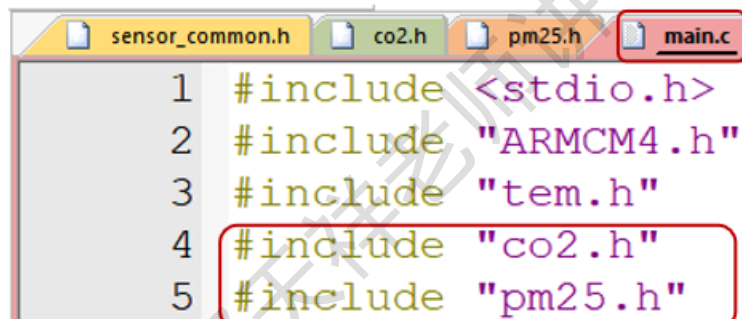
应用案例



```
1 #include "sensor_common.h"
2
3 QualityLevel GetCo2Level(void);
4 void DisplayCo2Level(QualityLevel lv);
```



```
1 #include "sensor_common.h"
2
3 QualityLevel GetPm25Level(void);
4 void DisplayPm25Level(QualityLevel lv);
```



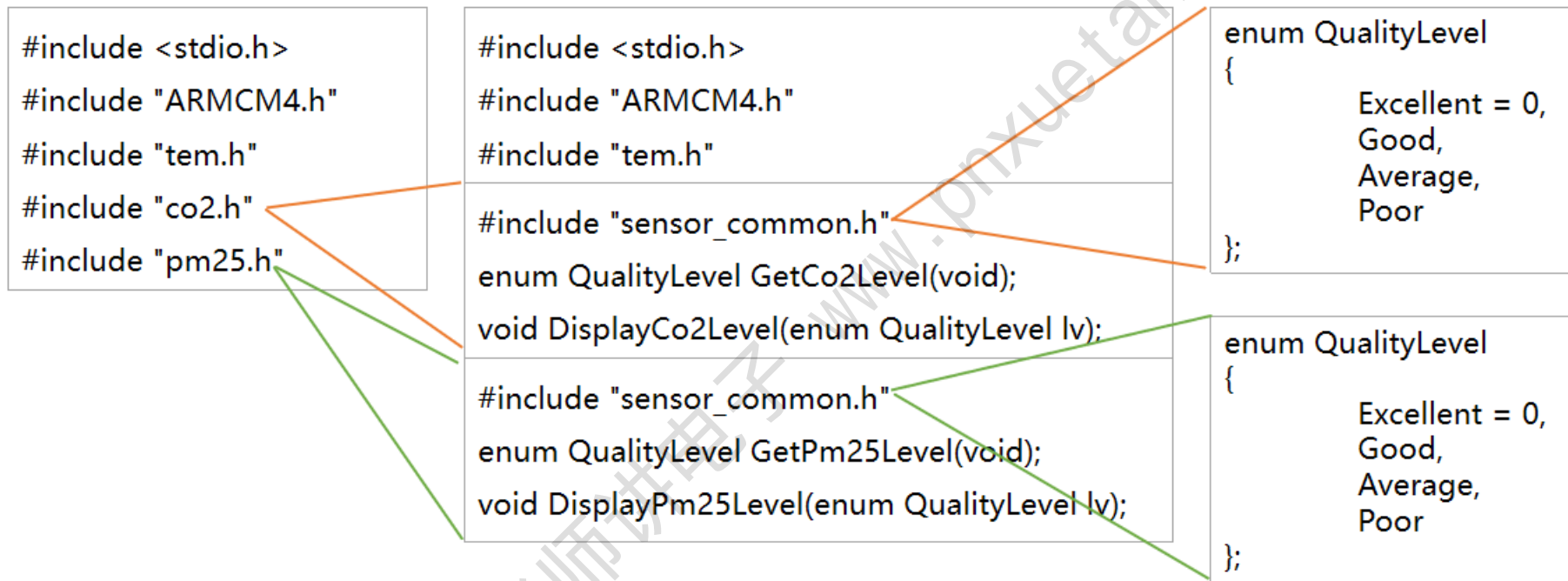
```
1 #include <stdio.h>
2 #include "ARMCM4.h"
3 #include "tem.h"
4 #include "co2.h"
5 #include "pm25.h"
```

应用案例

➤ 能否编译通过?

```
.\sensor\sensor_common.h(4): error: #101: "Excellent" has already been declared in the current scope
    Excellent,
.\sensor\sensor_common.h(5): error: #101: "Good" has already been declared in the current scope
    Good,
.\sensor\sensor_common.h(6): error: #101: "Average" has already been declared in the current scope
```

错误分析-展开后在一个C文件中有重复的自定义数据类型



条件编译

```
1 #ifndef SENSOR_COMMON_H
2 #define _SENSOR_COMMON_H_
3 typedef enum
4 {
5     Excellent,
6     Good,
7     Average,
8     Poor
9 }QualityLevel;
10
11 #endif
```

条件编译

```
#include <stdio.h>
#include "ARMCM4.h"
#include "tem.h"
```

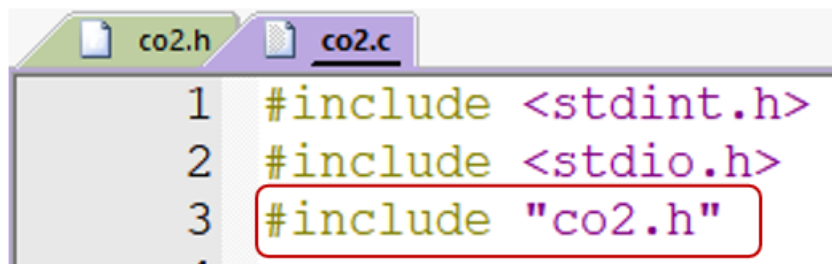
```
#include "sensor_common.h"
enum QualityLevel GetCo2Level(void);
void DisplayCo2Level(enum QualityLevel lv);
```

```
#include "sensor_common.h"
enum QualityLevel GetPm25Level(void);
void DisplayPm25Level(enum QualityLevel lv);
```

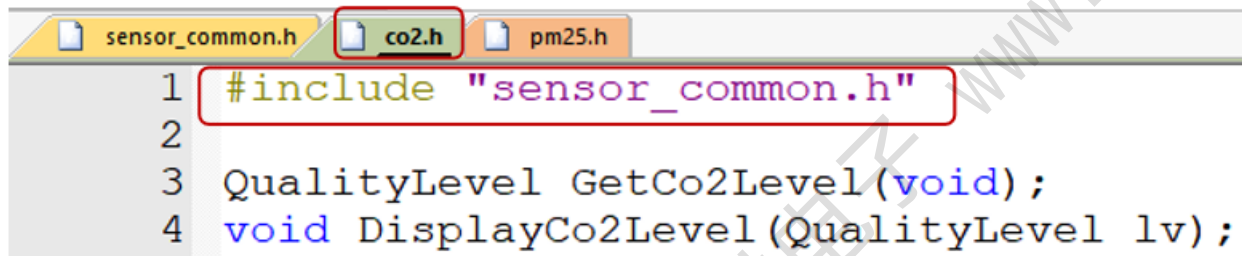
```
#ifndef _SENSOR_COMMON_H_
#define _SENSOR_COMMON_H_
enum QualityLevel
{
    Excellent = 0,
    Good,
    Average,
    Poor
};
#endif
```

```
#ifndef _SENSOR_COMMON_H_
#define _SENSOR_COMMON_H_
enum QualityLevel
{
    Excellent = 0,
    Good,
    Average,
    Poor
};
#endif
```

条件编译

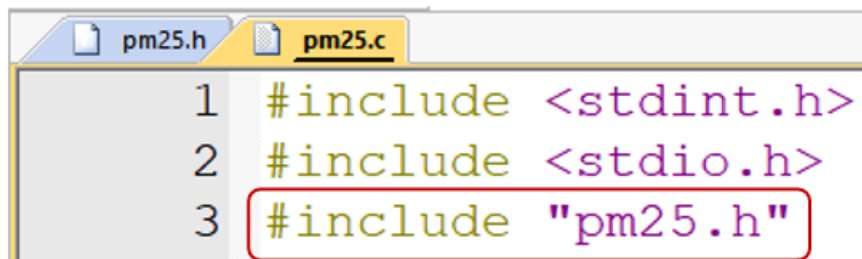


```
1 #include <stdint.h>
2 #include <stdio.h>
3 #include "co2.h"
```



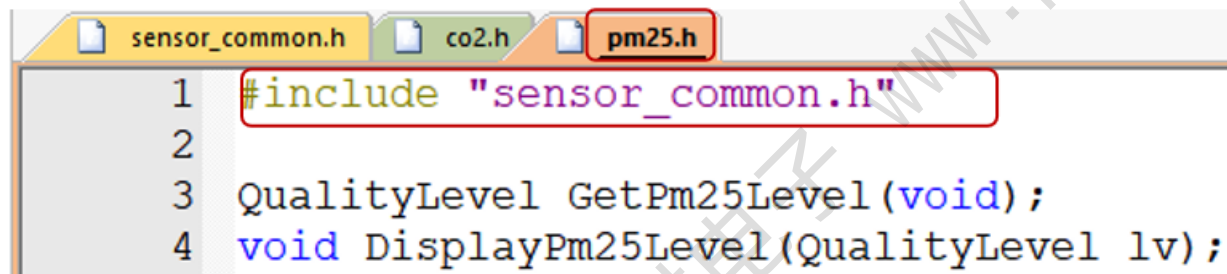
```
1 #include "sensor_common.h"
2
3 QualityLevel GetCo2Level(void);
4 void DisplayCo2Level(QualityLevel lv);
```

条件编译



The image shows a code editor window with two tabs: 'pm25.h' and 'pm25.c'. The 'pm25.c' tab is active. The code contains three lines: line 1 is '#include <stdint.h>', line 2 is '#include <stdio.h>', and line 3 is '#include "pm25.h"'. The third line is highlighted with a red rectangular box.

```
1 #include <stdint.h>
2 #include <stdio.h>
3 #include "pm25.h"
```



The image shows a code editor window with three tabs: 'sensor_common.h', 'co2.h', and 'pm25.h'. The 'pm25.h' tab is active. The code contains four lines: line 1 is '#include "sensor_common.h"', line 2 is an empty line, line 3 is 'QualityLevel GetPm25Level(void);', and line 4 is 'void DisplayPm25Level(QualityLevel lv);'. The first line is highlighted with a red rectangular box.

```
1 #include "sensor_common.h"
2
3 QualityLevel GetPm25Level(void);
4 void DisplayPm25Level(QualityLevel lv);
```

1.多个C文件包含相同的头文件，该头文件有条件编译宏控制，是否会在每个C文件中展开？

2.在多个C文件中有相同的自定义数据类型，是否有问题？

3.如果在该头文件中定义了一个变量，是否有问题？

1.多个C文件包含相同的头文件，该头文件有条件编译宏控制，是否会在每个C文件中展开？

➤ **#define:**

不管是在某个函数内，还是在所有函数之外，作用域都是从定义开始直到整个文件结尾。

➤ **typedef:**

如果放在所有函数之外，它的作用域就是从它定义开始直到文件尾；

如果放在某个函数内，定义域就是从定义开始直到该函数结尾；

➤ 不管是typedef还是define，其作用域都不会扩展到别的文件，即使是同一个程序的不同文件，也不能互相使用。

1.多个C文件包含相同的头文件，该头文件有条件编译宏控制，是否会在每个C文件中展开？

2.在多个C文件中有相同的自定义数据类型，是否有问题？

3.如果在该头文件中定义了一个变量，是否有问题？

```
pm25.h  pm25.c  sensor_common.h
1 #ifndef _SENSOR_COMMON_H_
2 #define _SENSOR_COMMON_H_
3
4 int a;
5
6 typedef enum
7 {
8     Excellent,
9     Good,
10    Average,
11    Poor
12 }QualityLevel;
13
14 #endif
```

linking...

```
.\Objects\template.axf: Error: L6200E: Symbol a multiply defined (by co2.o and main.o).
.\Objects\template.axf: Error: L6200E: Symbol a multiply defined (by pm25.o and main.o).
```

THANK YOU!