

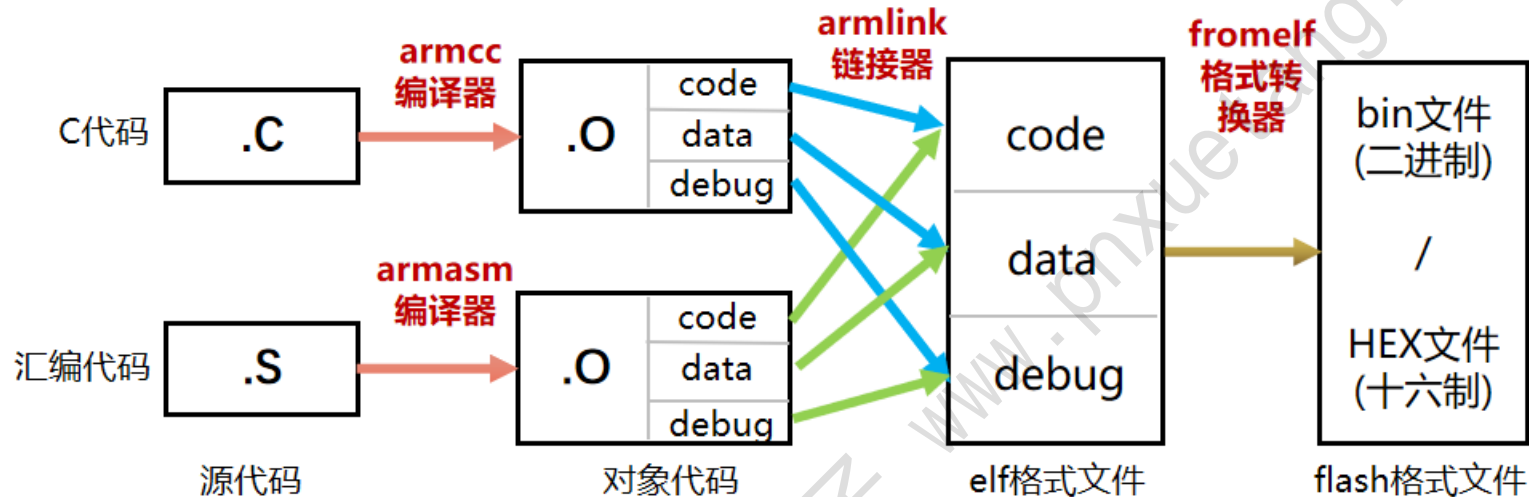
# 嵌入式C语言之- 函数的声明和定义

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

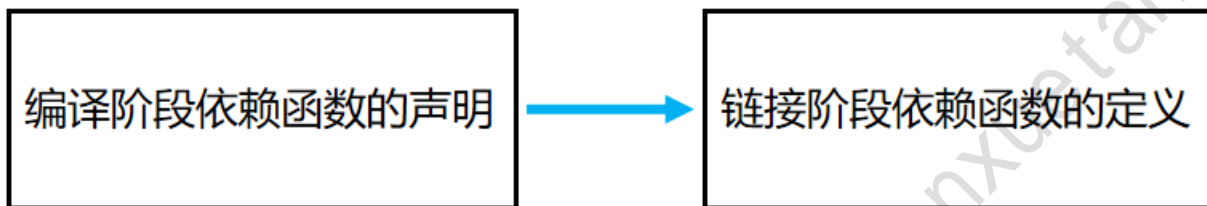
```
void AFunc(void);
```

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;
}
```

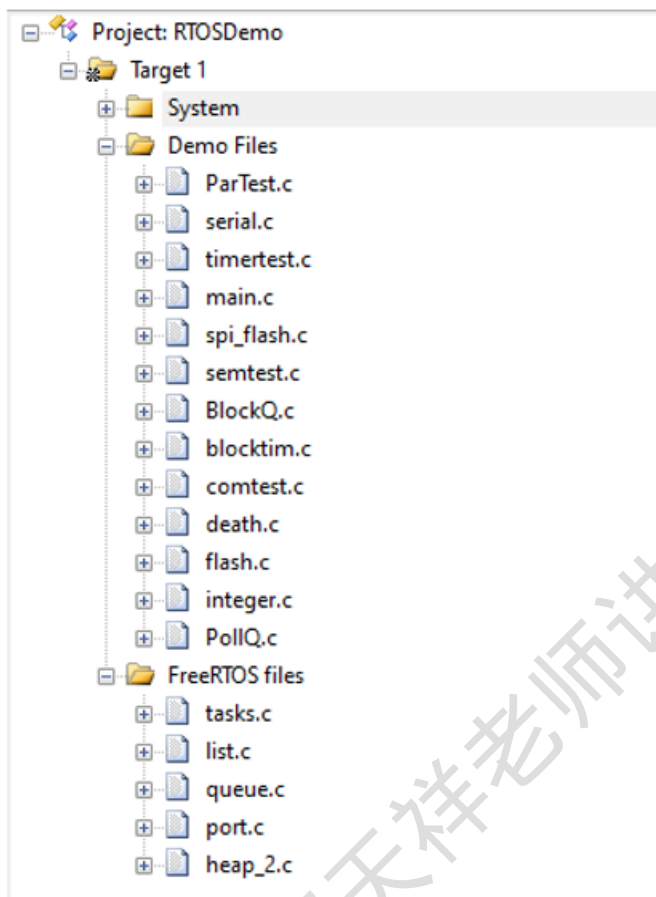
```
void AFunc(void);

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

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

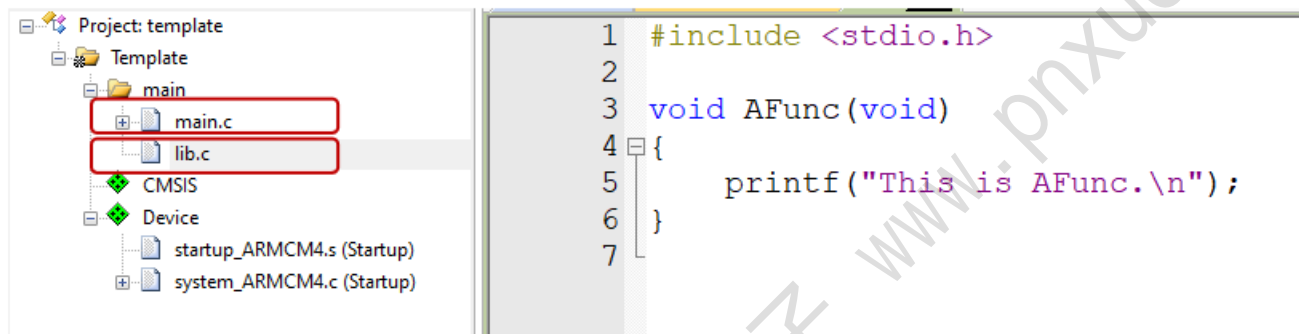
- 没有明确的标准，每个公司的规范都不尽相同，建议采用第一种方式，代码会更简洁，便于阅读。

# 为什么多源文件开发?

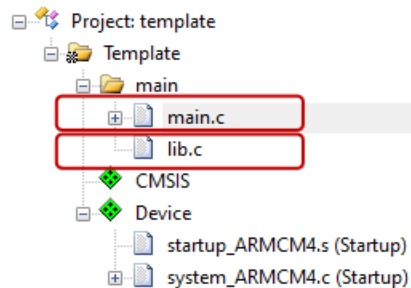


1. 便于团队合作，提升效率，同时开发一个源文件，会导致编写的代码冲突；
2. 代码结构清晰，每个源文件对应一个功能模块，便于维护和阅读。

# 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.

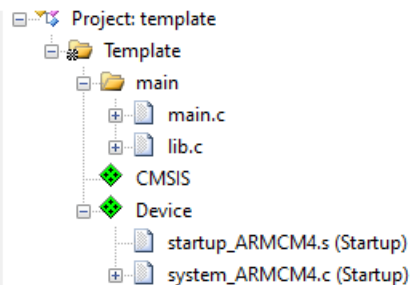
# 在不同C文件中，函数的定义和声明

- 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



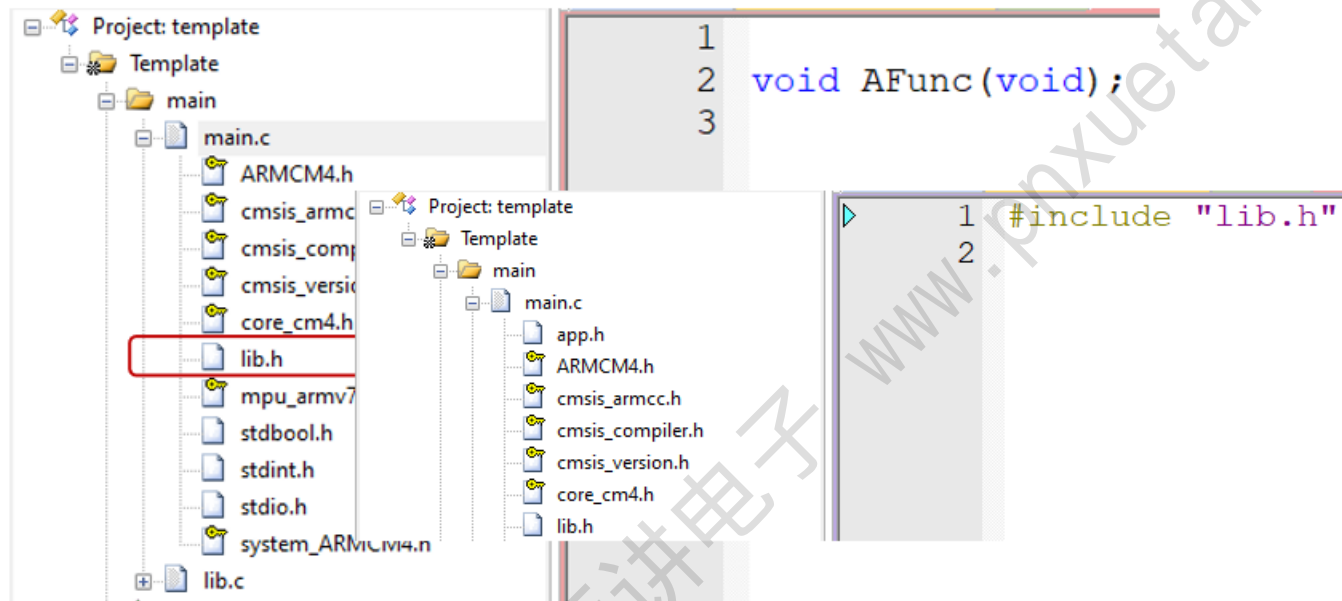
```
20 }
21 int ferror(FILE *f)
22 {
23     /* Your implementation of ferror */
24     return EOF;
25 }
26
27 void _ttywrch(int ch)
28 {
29     fputc(ch, &__stdout);
30 }
31
32 void _sys_exit(int return_code)
33 {
34     while (1);    /* endless loop */
35 }
36
37 void AFunc(void);
38
39 int main(void)
40 {
41     AFunc();
42     return 0;
43 }
```

在调用的C文件中，提前声明，可以解决编译警告

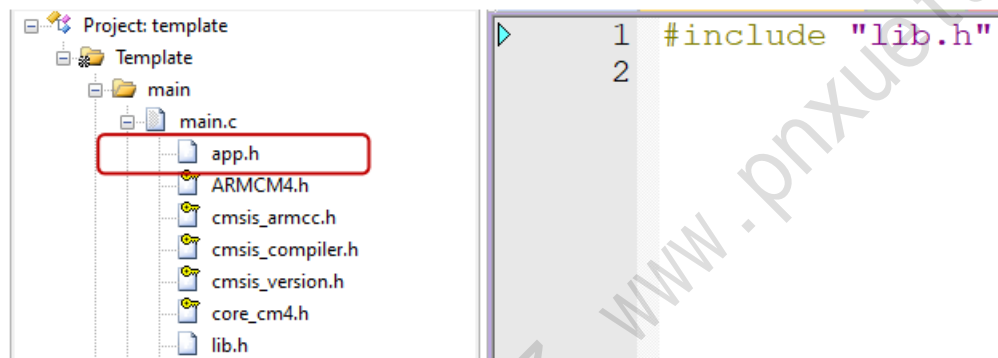
## 在不同C文件中，函数的定义和声明，更规范用法

```
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 extern void AFunc(void);
37
38 int main(void)
39 {
40     AFunc();
41     return 0;
42 }
```

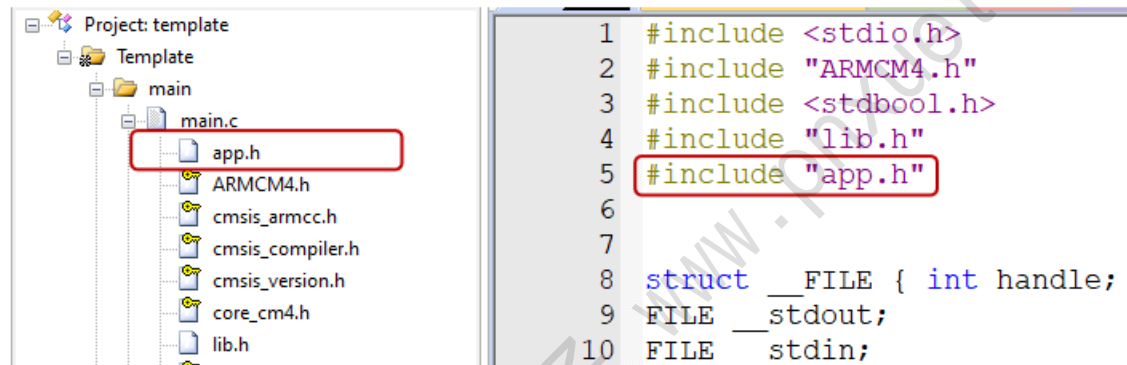
# 在不同C文件中，函数的定义和声明，最规范用法



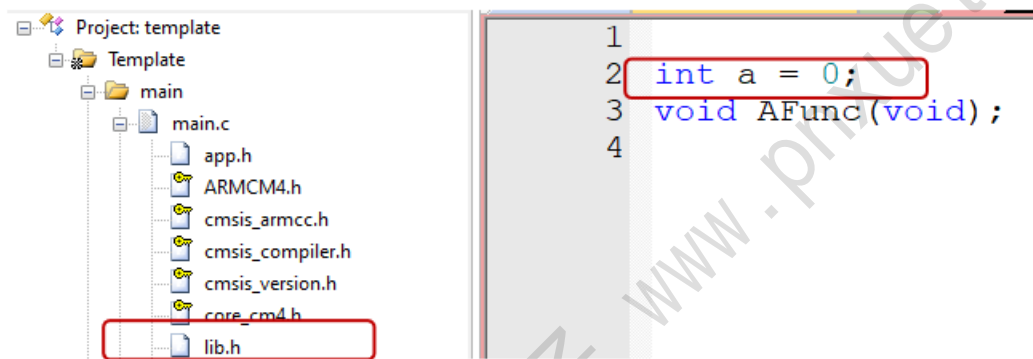
# 在不同C文件中，函数的定义和声明



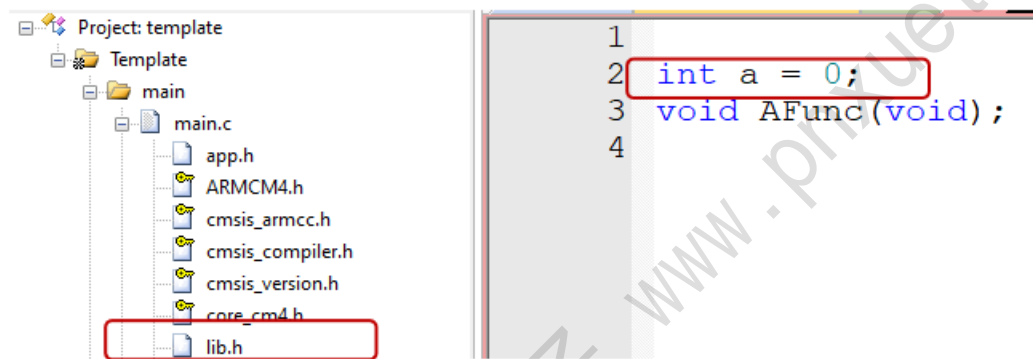
# 在不同C文件中，函数的定义和声明



# 在不同C文件中，函数的定义和声明

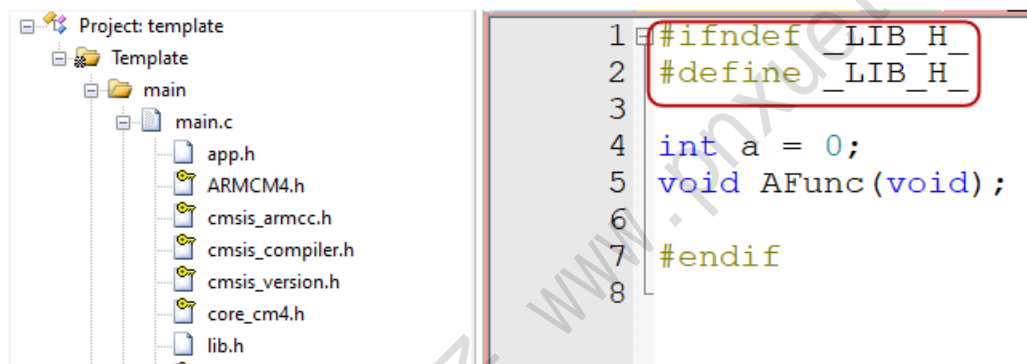


# 在不同C文件中，函数的定义和声明



lib.h(2): error: #148: variable "a" has already been initialized

# 在不同C文件中，函数的定义和声明

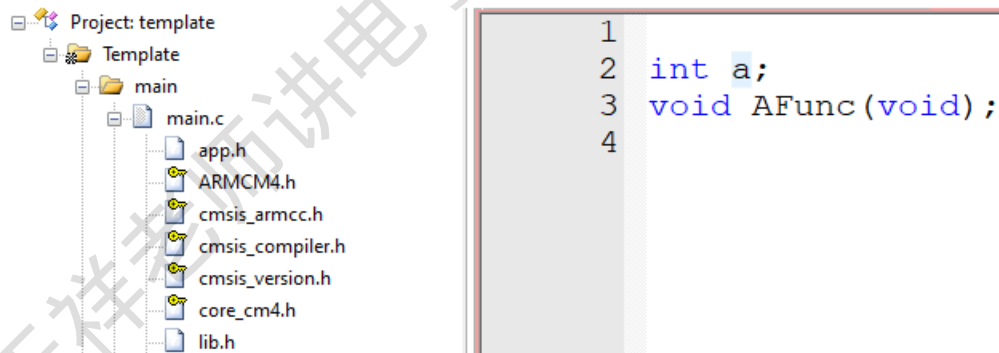


The screenshot shows an IDE with a project structure on the left and a C file editor on the right. The project structure is as follows:

- Project: template
  - Template
    - main
      - main.c
      - app.h
      - ARMCM4.h
      - cmsis\_armcc.h
      - cmsis\_compiler.h
      - cmsis\_version.h
      - core\_cm4.h
      - lib.h

The C file editor shows the following code:

```
1 #ifndef _LIB_H_
2 #define _LIB_H_
3
4 int a = 0;
5 void AFunc(void);
6
7 #endif
8
```



The screenshot shows the same IDE with the same project structure. The C file editor shows the following code:

```
1
2 int a;
3 void AFunc(void);
4
```



**THANK YOU!**