

千锋嵌入式学院C语言培训

—预处理

Author:Richard.zhang

源自清华 值得信赖

Section 1: 宏定义

定义一个宏

#define 标识符 字符串

```
#define G 9.8
```

```
#define PI 3.1415926
```

```
#define INTEGER int
```

```
#define VAR a
```

需要注意的问题

- ▶ 宏展开只是简单字符串替换，不做正确性检查，这一点应该特别注意。
- ▶ 宏的有效范围是从定义出开始到文件的结尾。若想提前取消宏，则可以使用
 `#undef` 宏名
取消宏。

容易混淆的一点

```
#define G 1  
int main()  
{  
    .....  
}  
#undef G  
int f(){ ..... }
```

带参数的宏定义

`#define` 宏名(参数表) 字符串

```
#define max(a,b) a>b?a:b
```

```
/* 此表达式得出a和b的较大值 */
```

```
int c = max(a,b);
```

```
#define MAX(a, b) ((a)>(b)?(a):(b))
```

就像函数调用一样，把两个实参分别替换到宏定义中形参a和b的位置。注意这种函数式宏定义和真正的函数调用有什么不同：

- 1、函数式宏定义参数没有类型，预处理器只负责做形式上的替换，而不做参数类型检查，所以传参时要格外小心。
- 2、调用真正函数的代码和调用函数式宏定义的代码编译生成的指令不同。如果MAX是个真正的函数，那么它的函数体return a > b ? a : b;要编译生成指令，代码中出现的每次调用也要编译生成传参指令和call指令。而如果MAX是个函数式宏定义，这个宏定义本身倒不必编译生成指令，但是代码中出现的每次调用编译生成的指令都相当于一个函数体，而不是简单的几条传参指令和call指令。所以，使用函数式宏定义编译生成的目标文件会比较大。

- 3、定义这种宏要格外小心，如果上面的定义写成`#define MAX(a, b) (a>b?a:b)`，就容易出现错误的运算。
- 4、调用函数时先求实参表达式的值再传给形参，如果实参表达式有Side Effect，那么这些Side Effect只发生一次。例如`MAX(++a, ++b)`，如果MAX是个真正的函数，a和b只增加一次。但如果MAX是上面那样的宏定义，则要展开成`k = ((++a)>(++b)?(++a):(++b))`，a和b就不一定是增加一次还是两次了。

带参数的宏定义

当宏定义包括多行的时候应采用如下写法：

```
#define max(a,b,c) { \  
    if(a > b) \  
        c = a - b; \  
    else \  
        c = b - a; \  
}
```

其中 “\”代表连接两行为一个宏字符串内容

宏的副作用

定义如下宏

```
#define test(a,b) a * b
```

在程序使用该宏

```
test(a + b, a - b);
```

该宏被展开成: $a + b * a - b$;

和我们期望的运算顺序不符

正确的写法是:

```
#define test(a,b) (a) * (b)
```

宏的副作用

定义如下宏

```
#define S (a,b) a + b
```

注意，宏名和参数列表之间有一个空格，这个时候在使用宏的时候会出现问题

S(a,b)会被展开为：

(a,b) a + b(a,b)

这个结果和我们预想的大相径庭，所以说在定义宏的时候一定要注意空格和\t之类的字符

一个不规范写法引发的错误

定义一个宏

```
#define f(); printf("hello"); printf("world");
```

使用此宏

```
if(a > b)
```

```
    f();
```

```
else
```

```
    printf("hello world\n");
```

编译的结果会怎样？

一个不规范写法引发的错误

正确的写法是：

```
if(a > b){ f(); }  
else printf("hello world\n");
```

宏展开的形式如下：

```
If(a > b){  
    printf("hello");  
    printf("world");  
}  
else  
    printf("hello world\n");
```

宏和函数的区别

- ▶ 宏是编译时的概念，函数是运行时的概念
- ▶ 宏在编译时发生，运行速度较快；函数在运行时发生，运行速度较慢
- ▶ 宏不会对参数进行类型检查，函数有严格的类型检查
- ▶ 宏不分配存储单元，函数需要分配存储单元
- ▶ 宏会使代码体积变大，函数不会
- ▶ 宏不可以调用自身，函数可以

宏的使用实例

(1)单片机和嵌入式系统编程中经常出现的死循环:

```
#define panic(); while(1);
```

```
#define die(); for(;;);
```

(2)空指针NULL

```
#define NULL (void *)0
```

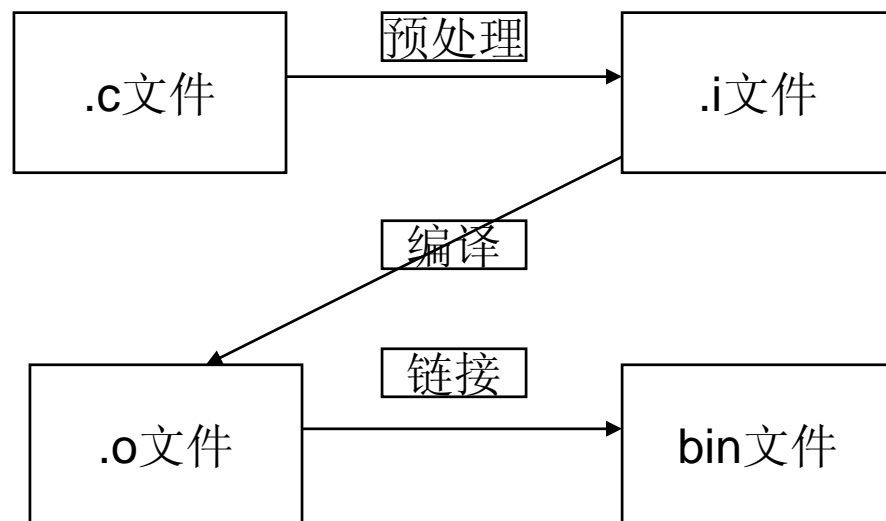

inline关键字

inline和宏的区别：

- (1)宏展开是在预处理发生的，而inline函数展开是在编译发生的。
- (2)inline函数是一种函数，会严格进行参数类型检查，因此可以避免宏的一些副作用。
- (3)Inline函数不能有过于复杂的语句，而宏对此没有要求。
- (4)Inline函数不能保证一定展开，而宏能保证一定被展开

Section 2: 文件包含

预处理的概念



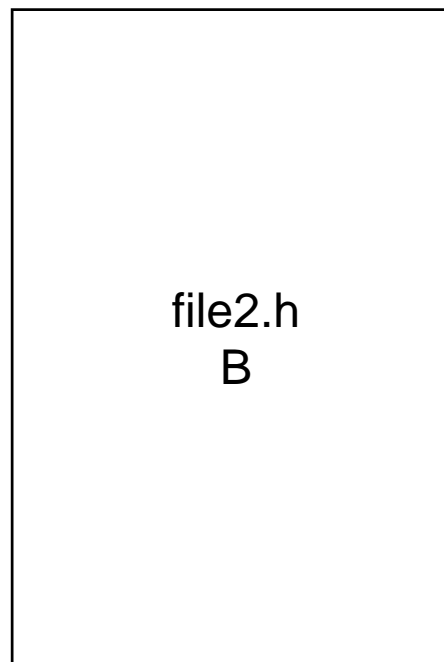
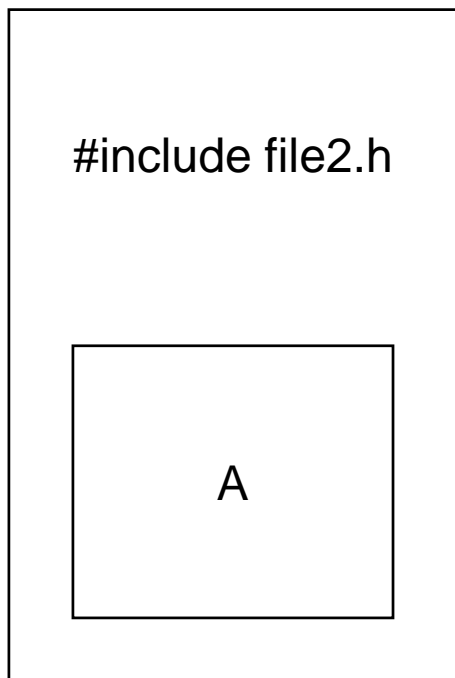
文件包含

#include的两种方式:

- ▶ #include <file.h>
- ▶ #include "file.h"

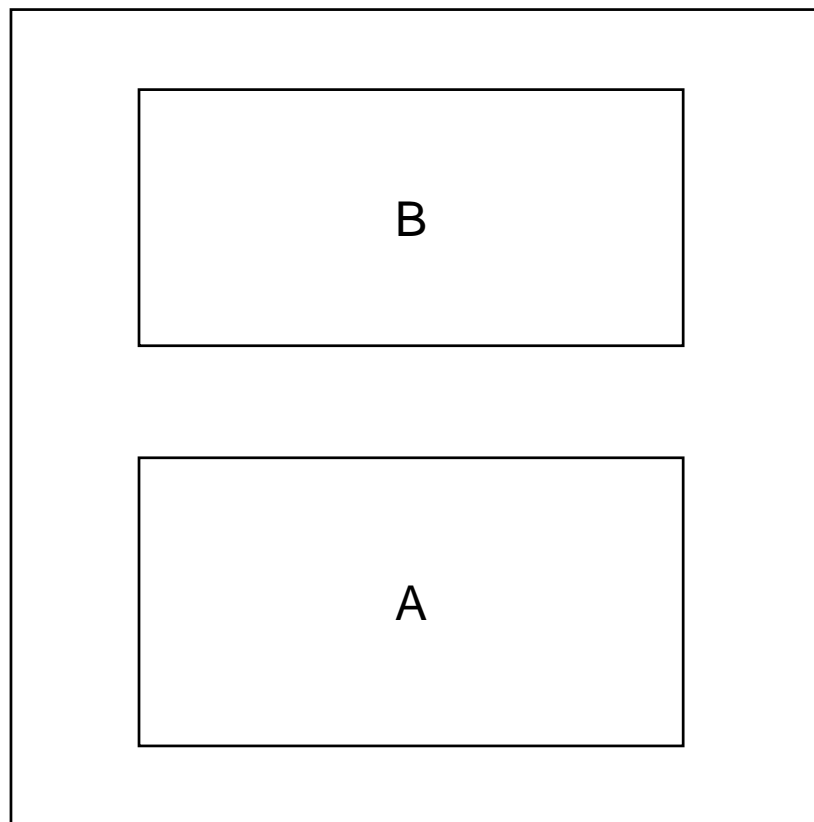
文件包含机制

file1.c



文件包含机制

file1.c



需要注意的问题

- ▶ (1)包含是在源程序级别的，而不是目标文件级别的，也就是说预处理不会干涉编译级的处理
- ▶ (2)注意包含顺序的问题

需要注意的问题

- ▶ (3)嵌套文件包含

- ▶ (4)注意包含文件的一个陷阱

例如：file1.h中定义a变量int a = 1;file2.h中也定义了a变量int a = 0;file3.c包含file1.h和file2.h时会怎么样？如果把int a = 1;和int a = 0;都改成int a;又会怎么样呢？

Section 3: 条件编译

条件编译

- ▶ 条件编译形式1

```
#ifdef 标识符
    程序段1
#else
    程序段2
#endif
```

条件编译

```
#define G  
#ifdef G  
    printf("the symbol G is defined\n");  
#else  
    printf("the symbol G is not  
    defined\n");
```

编译此程序段，比较注释掉第一行后编译的运行结果。

条件编译

- ▶ 条件编译形式2
#ifndef 标识符
 程序段1
#else
 程序段2
#endif

条件编译

▶ 条件编译形式3

#if 表达式

程序段1

#else

程序段2

#endif

条件编译应用

(1) 调试开关

```
#define DEBUG
#ifdef DEBUG
#define DEBUG_PRINT1(arg1) printf(arg1)
#define DEBUG_PRINT2(arg1,arg2) printf(arg1,arg2)
#define DEBUG_PRINT2(arg1,arg2,arg3) printf(arg1,arg2,arg3)
#else
#define DEBUG_PRINT1(arg1)
#define DEBUG_PRINT2(arg1,arg2)
#define DEBUG_PRINT2(arg1,arg2,arg3)
#endif
```


条件编译应用

(2) 便于确定哪些头文件没有编译

```
#ifndef FILE_H_
```

```
#define FILE_H_
```

```
.....
```

```
#endif
```

条件编译头文件file.h，防止由于不小心对于一些函数或者变量的重复定义

typedef关键字

typedef 关键字用于自定义一个类型。

```
typedef long size_t;  
typedef int type_t;
```

对于简单类型来讲，自定义类型有点类似于宏替换。

```
size_t a;  
long a;
```

typedef关键字

typedef 关键字与宏的区别:

```
typedef int type_t;
```

```
#define INT int;
```

```
type_t a;
```

```
INT b;
```

```
typedef int * pointer;
```

```
#define Pointer int *
```

```
Pointer p, q;
```

```
pointer p,q;
```

typedef关键字

对于复杂类型:

```
typedef int (*ARR)[10];
```

```
ARR arr_p;
```

```
typedef int (*FUNC)(int , int);
```

```
FUNC func_p;
```

typedef关键字

使用typedef关键字简化函数指针的声明

```
void (*signal(int signo, void (*func)(int)))(int);
```

转换为

```
typedef void (*FUNC)(int);
```

```
FUNC signal(int signo , FUNC func);
```

本次课程知识点总结

文件包含: 1 C语言中预处理的概念

2 文件包含的机制

宏定义: 1 宏定义的概念和使用

2 带参数的宏的定义

3 宏的副作用

4 宏和函数的区别

5 inline关键字的用法

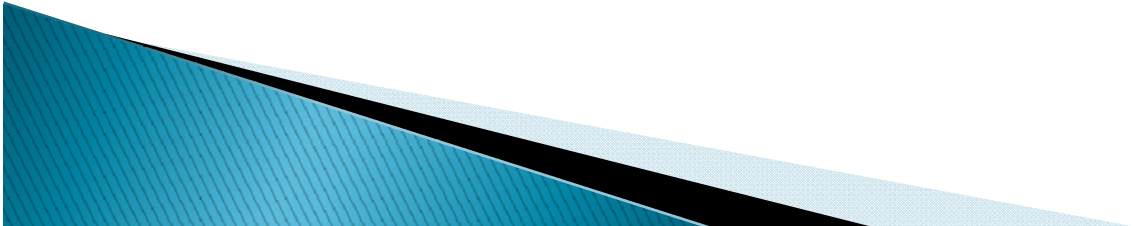
6 inline关键字和宏的区别

条件编译: 1 条件编译的三种形式

2 条件编译的两种应用举例

3 条件编译与编译选项

Thank you



源自清华 值得信赖