



西北工业大学  
NORTHWESTERN POLYTECHNICAL UNIVERSITY

---

# C程序设计 Programming in C



**1011014**

---

主讲：姜学锋，计算机学院

## 编程任务的自动化

- ◆ 3、带参数的宏定义
- ◆ 4、编译器预定义宏

## 5.1.2 带参数的宏定义

---

- ▶ 带参数的宏定义的命令形式为：

```
#define 宏名(参数表) 字符文本
```

- ▶ 带参数的宏的引用形式为：

```
宏名(引用参数表)
```

## 5.1.2 带参数的宏定义

---

- ▶ 其中：
- ▶ ①参数表允许多个参数，用逗号分隔，称为形式参数（不同于函数的形参概念）。
- ▶ ②字符文本中包含所指定的参数文本，出现次序和数目没有任何限制。
- ▶ ③引用参数表与宏定义的形式参数要求一一对应。

### 5.1.2 带参数的宏定义

---

- ▶ 预处理时，预处理器先将宏引用的引用参数文本对应地替换宏定义字符文本中的参数，接下来进行宏替换，然后再进行编译。

## 5.1.2 带参数的宏定义

---

- ▶ 例如有宏定义

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

- ▶ 形式参数按顺序为a和b，而程序代码：

```
L=max(x-y,x+y); //max宏引用
```

- ▶ 引用参数文本按顺序为x-y和x+y。

## 5.1.2 带参数的宏定义

---

- ▶ 先将引用参数文本对应地替换字符文本中的参数，字符文本中其他内容不变，则字符文本置换为：

```
((x-y) > (x+y)) ? (x-y) : (x+y))
```

- ▶ 因此预处理时宏替换为：

```
L=((x-y) > (x+y)) ? (x-y) : (x+y))
```

### 5.1.2 带参数的宏定义

---

- ▶ 使用带参数的宏定义需要注意：
- ▶ （1）宏名与“(参数表)”之间不能有空白符，否则命令形式会被理解为不带参数的宏定义，而“(参数表)”是字符文本的一部分。



## 5.1.2 带参数的宏定义

---

▶ 例如：

```
#define AREA (r) PI*r*r
```

▶ AREA为不带参数的宏，“(r) PI\*r\*r”组成了它的字符文本，因此：

```
S=AREA(x); //AREA宏引用
```

▶ 展开为：

```
S=(r) PI*r*r(x)
```

▶ 显然是不对的。

## 5.1.2 带参数的宏定义

---

- ▶ (2) 字符文本中的参数必须是由各种符号、空白符分隔出来的独立字符文本，例如：

```
#define MSET1(arg) x=Aarg+2;  
#define MSET2(arg) x=A+arg;  
MSET1(1) //宏替换为 x=Aarg+2; arg没有被替换  
MSET2(1) //宏替换为 x=A+1; arg已替换
```

- ▶ Aarg字符文本不会进行arg参数替换，因为Aarg是一个不可分割的整体。

### 5.1.2 带参数的宏定义

---

- ▶ (3) 引用参数文本替换字符文本中的参数时，只是简单地做文本替换，某些表达式的宏定义中，这种简单处理可能会得到不符合原意的替换结果。

## 5.1.2 带参数的宏定义

---

▶ 例如：

```
#define POWER(a) a*a //计算a的平方
```

▶ 如果宏引用为：

```
S=POWER(x); //宏替换为 S=x*x
```

▶ 是正确的，但如果宏引用为：

```
S=POWER(x+y); //宏替换为 S=x+y*x+y
```

▶ 得到的宏扩展“ $x+y*x+y$ ”显然与“ $(x+y)*(x+y)$ ”原意不符。

## 5.1.2 带参数的宏定义

---

- ▶ 解决这个问题有两种方法：
- ▶ 一是给引用参数文本加上括号，例如：

```
S=POWER((x+y)); //宏替换为 S=(x+y)*(x+y)
```

## 5.1.2 带参数的宏定义

---

- ▶ 二是在宏定义时给字符文本中的参数加上括号，例如：

```
#define POWER(a) (a)*(a) //计算a的平方  
S=POWER(x+y); //宏替换为 S=(x+y)*(x+y)
```

- ▶ 实际编程中，第二种方法更稳妥。

### 5.1.2 带参数的宏定义

---

- ▶ (4) 无论是带参数的宏定义或是不带参数的宏定义，均可以使用行连接符“\”得到多行宏定义，进而得到具有复杂功能的宏。

## 5.1.2 带参数的宏定义

---

► 例如：

```
1 #define PRINTSTAR(n) { \  
2     int i,j; \  
3     for(i=1;i<=n;i++) { \  
4         for (j=1;j<=i;j++) \  
5             printf("*"); \  
6         printf("\n"); \  
7     } \  
8 }
```



## 5.1.2 带参数的宏定义

---

► 那么PRINTSTAR(5)宏引用的结果实际上是如下程序代码：

```
1 {  
2     int i,j;  
3     for(i=1;i<=5;i++) {  
4         for (j=1;j<=i;j++)  
5             printf("*");  
6         printf("\n");  
7     }  
8 }
```

### 5.1.2 带参数的宏定义

---

- ▶ 这里外加一对花括号“{ }”的目的是形成一个复合语句，局部区域变量，它们与外部不会有任何冲突。
- ▶ 需要注意，宏定义的最后要连接一个空行，这样宏替换时才会有相应的换行。

## 5.1.2 带参数的宏定义

---

► PRINTSTAR(5)的运行结果为：

```
*  
**  
***  
****  
*****
```

## 5.1.2 带参数的宏定义

---

- ▶ 可以用不同的参数引用宏PRINTSTAR，得到数目不同的星号输出，例如：

```
PRINTSTAR(5) //宏替换为一段程序代码  
PRINTSTAR(8) //宏替换为一段程序代码  
PRINTSTAR(10) //宏替换为一段程序代码
```

- ▶ 善于利用宏定义，可以实现程序的简化。

### 5.1.2 带参数的宏定义

---

- ▶ 带参数的宏定义的引用与函数调用在语法上比较相似，例如在调用函数时在函数名后的括号内写实参，要求实参与形参的顺序对应和数目相等。但它们基本含义不同，主要区别是：
- ▶ （1）函数调用时会先计算实参表达式的值，然后参数值传递给形参，程序指令会转到函数内部开始执行。而带参数的宏定义只是参数文本替换，不存在计算实参、参数传递、跳转执行等。

### 5.1.2 带参数的宏定义

---

- ▶ (2) 函数调用是在程序运行时执行的，它会为形式参数分配临时的内存单元。而宏在预处理阶段替换，不会为形式参数分配内存单元，而且也没有返回和返回值的概念。
- ▶ (3) 函数调用对实参和形参都要定义类型，且要求二者的类型一致，如果不一致，会进行类型转换。而宏定义不存在类型问题，它的形式参数和引用参数都只是一个文本记号，宏替换时进行文本置换。

## 5.1.2 带参数的宏定义

---

- ▶ (4) 无参数函数调用必须包含括号，无参数宏定义引用时不需要括号。例如：

```
#define PI 3.1415926 //宏定义  
int fun(); //函数原型  
x=fun(); //函数调用  
x=PI; //宏引用
```

### 5.1.2 带参数的宏定义

---

- ▶ (5) 每一次宏引用，宏替换后都会使源程序增长，相当于将宏定义的字符文本“粘贴”到源程序中一次，而函数调用代码是复用的。宏替换会占用编译时间，函数调用则会占用运行时间。



### 5.1.2 带参数的宏定义

---

- ▶ (6) 宏定义与前面讲的内联函数非常相似。两者区别在于：宏是由预处理器对宏进行替换，它是在代码处不加任何检验的简单替换；而内联函数是通过编译器来实现的，它有函数的特性，只是在需要用到的时候，内联函数像宏一样地展开，取消了函数的参数入栈，减少了调用的开销。内联函数要做参数类型检查，这是内联函数跟宏相比的优势。

## 5.1.2 带参数的宏定义

---



### 【例5.2】

---

宏引用和函数调用的区别。


## 5.1.2 带参数的宏定义

### 例5.2

```
1 #include <stdio.h>
2 int M1(int y)
3 {
4     return((y)*(y));
5 }
6 #define M2(y) ((y)*(y))
7 int main()
8 {
9     int i,j;
10    for(i=1,j=1;i<=5;i++) printf("%d ",M1(j++)); //函数调用处理
11    printf("\n");
12    for (i=1,j=1;i<=5;i++) printf("%d ",M2(j++)); //宏引用处理
13    printf("\n");
14    return 0;
15 }
```

## 5.1.2 带参数的宏定义

### 例5.2



```
1 #include <stdio.h>
2 int M1(int n)
3 {
4     int i;
5     for (i = 1; i <= n; i++)
6         printf("%d ", M1(j++)); //函数调用处理
7     printf("%d ", M2(j++)); //宏引用处理
8 }
```

1 4 9 16 25  
1 9 25 49 81

## 5.1.2 带参数的宏定义

---

例5.2

```
1 #include <stdio.h>
2 int M1(int y)
3 {
4     return((y)*(y));
5 }
6 #define M2(y) ((y)*(y))
```

可以看出函数调用和宏引用在形式上相似，在本质上是完全不同的。

### 5.1.3 #和##预处理运算

---

- ▶ C语言标准为预处理命令定义了两个运算符：#和##，它们在预处理时被执行。

### 5.1.3 #和##预处理运算

---

- ▶ #运算符的作用是文本参数“字符串化”，即出现在宏定义字符文本中的#把跟在后面的参数转换成一个C语言字符串常量。  
例如：

```
#define PRINT_MSG1(x) printf(#x);  
#define PRINT_MSG2(x) printf(x);  
PRINT_MSG1(Hello World); //正确，宏替换为 printf("Hello World");  
PRINT_MSG1("Hello World");  
//正确，宏替换为 printf("\\"Hello World\\"");  
PRINT_MSG2(Hello World); //错误，宏替换为 printf(Hello World);  
PRINT_MSG2("Hello World");  
//正确，宏替换为 printf("Hello World");
```

### 5.1.3 #和##预处理运算

---

- ▶ 简单来说，#参数的作用就是对这个参数替换后，再加双引号括起来，变为“参数”。



### 5.1.3 #和##预处理运算

---

- ▶ ##运算符的作用是将两个字符文本连接成一个字符文本，如果其中一个字符文本是宏定义的参数，连接会在参数替换后发生。例如：

```
#define SET1(arg) A##arg=arg;  
#define SET2(arg) Aarg=arg;  
SET1(1); //宏替换为 A1=1;  
SET2(1); //宏替换为 Aarg=1;
```

- ▶ A字符与##arg参数连接在一起形成了A1，而对于Aarg字符文本，不会进行arg替换。

## 5.1.4 预定义宏

---

- ▶ C语言标准中预先定义了一些有用的符号常量，这些符号常量主要是编译信息。

### 5.1.4 预定义宏

表5-1 标准预定义符号常量

符号常量	类型	说明
__DATE__	字符串常量	编译程序日期（形式为“MM DD YYYY”，例如“May 4 2006”）
__TIME__	字符串常量	编译程序时间（形式为“hh:mm:ss”，例如“10:20:05”）
__FILE__	字符串常量	编译程序文件名
__LINE__	int型常量	当前源代码的行号
__STDC__	int型常量	ANSI C标志，若为1说明此程序兼容ANSI C标准

其中“\_\_”为两个下划线，\_\_DATE\_\_和\_\_TIME\_\_用于指明程序编译的时间，\_\_FILE\_\_和\_\_LINE\_\_用于调试目的，\_\_STDC\_\_检测编译系统是否支持C语言标准。

## 5.1.4 预定义宏

---

例5.51

```
1 #include <stdio.h>
2 int main()
3 {
4     printf("%s,%s,%s,%d\n",__DATE__,__TIME__,__FILE__,__LINE
__);
5     return 0;
6 }
```

某次运行结果为：

Oct 12 2010,21:49:41,D:\DEVSHOP\CH0551.c,4

**CP** 程序设计