

C语言与程序设计

The C Programming Language



第6章 编译预处理

华中科技大学计算机学院

毛伏兵



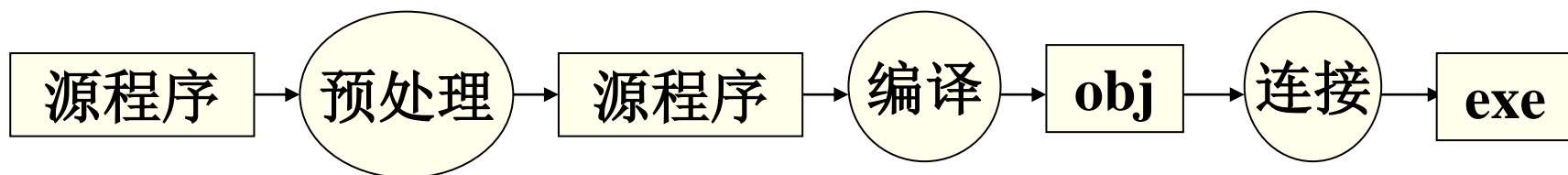
本章重点

■宏定义 `#define`

无参宏定义

带参宏定义 ✓

编译预处理



编译预处理：对源程序进行编译之前所作的工作，它由预处理程序负责完成。编译时，系统将自动引用预处理程序对源程序中的预处理指令进行处理。

预处理指令：以“#”号开始的指令。



6.1 文件包含#include

用指定文件的内容取代该预处理指令行，有2种一般形式：

(1) **#include <文件名>**

在指定的标准目录下寻找被包含文件

(2) **#include "文件名"**

首先在用户当前目录中寻找被包含文件，
若找不到，再在指定的标准目录下寻找

6.2 宏定义#define

用一个标识符来表示一个字符串。

#define 标识符 字符串

宏名：被定义的标识符。

宏代换（宏展开）：在编译预处理时，用字符串去取代宏名。

预处理前

```
#define M (y*y+3*y)
void main(void)
{ int s,y;
  printf("Input a number: ");
  scanf("%d",&y);
  s=3*M+4*M+y*M;
  printf("s=%d\n",s);
}
```

预处理后

```
void main(void)
{ int s,y;
  printf("Input a number: ");
  scanf("%d",&y);
  s=3*(y*y+3*y)+4*(y*y+3*y)
    +y*(y*y+3*y);
  printf("s=%d\n",s);
}
```

6.3 带参数的宏定义

#define 标识符(标识符, 标识符, ..., 标识符) 字符串

宏名

形式参数

宏调用：给出实参

宏展开：（1）用字符串替换宏，
（2）用实参去替换形参



例 定义计算 x^2 的宏

```
#define SQ(x) ((x)*(x))
```

宏调用: **SQ(a+1)**

宏展开: **((a+1) * (a+1))**

宏调用: **SQ(SQ(a))**

宏展开: **((((a)*(a))) * (((a)*(a))))**

为什么要这么多的括号？

考虑：`#define SQ(x) x*x`

宏调用：`SQ(a+b)`

宏展开：`a+b*a+b` /* 与 $(a+b)*(a+b)$ 不同 */

再考虑：`#define SQ(x) (x)*(x)`

宏调用：`27/SQ(3)`

宏展开：`27/(3)*(3)` /* 值27, 与 $27/3^2$ 不同 */

定义带参数的宏时，为了保证计算次序的正确性，表达式中的每个参数用括号括起来，整个表达式也用括号括起来。



注意：宏名和左括号之间不能有空格

#define SQ (x) ((x)*(x))

被认为是无参宏定义。

宏调用：**SQ(3)**

宏展开：**(x) ((x)*(x)) (3) /*显然错误的*/**



带参的宏虽被认为不安全，但还是很有价值

```
#define SQ(x) ((x)*(x))
```

宏调用：SQ(++a)

宏展开：((++a)*(++a)) /*a加2次

如是函数调用，将不会有问题 */

- 宏节省了函数调用的开销，程序运行速度更快，形式参数不分配内存单元，不必作类型说明。但是，宏展开后使源程序增长。
- 宏比较适合于经常使用的简短表达式，以及小的可重复的代码段；当任务比较复杂，需要多行代码才能实现时，或者要求程序越小越好时，就应该使用函数。



6.4 取消宏定义#undef

终止宏名的作用域，形式为：

#undef 标识符

何时使用 #undef 指令？

- 防止宏名的冲突

```
#include "everything.h"
```

```
#undef SIZE /*everything.h中定义了SIZE，就取消它；  
           否则该指令不起作用*/
```

```
#define SIZE 100
```

- 保证调用的是一个实际函数而不是宏

```
#undef getchar
```

```
int getchar(void) {...}
```



6.5 条件编译

- 预处理程序提供了条件编译指令，用于在预处理中进行条件控制，根据所求条件的值有选择地包含不同的程序部分，因而产生不同的目标代码。这对于程序的移植和调试是很有用的。对源程序的各部分有选择地进行编译称为条件编译。
- 条件编译指令三种形式，控制流与if-else语句类似。如表6.1所示。



例 利用R计算圆或正方形的面积

预处理前

```
#define R
void main(void)
{
    float c, s;
    printf("input a number: ");
    scanf("%f", &c);
    #ifdef R
    s=3.14159*c*c;
    printf("%f\n", r);
    #else
    s=c*c;
    printf("%f\n", s);
    #endif
}
```

预处理后

```
void main(void)
{
    float c, s;
    printf("input a number: ");
    scanf("%f", &c);
    s=3.14159*c*c;
    printf("%f\n", s);
}
```

生成的目标程序较短



例 利用R计算圆或正方形的面积

预处理前

```
void main(void)
{
    float c, s;
    printf("input a number: ");
    scanf("%f", &c);
    #ifdef R
    s=3.14159*c*c;
    printf("%f\n", s);
    #else
    s=c*c;
    printf("%f\n", s);
    #endif
}
```

预处理后

```
void main(void)
{
    float c, s;
    printf("input a number: ");
    scanf("%f", &c);
    s=c*c;
    printf("%f\n", s);
}
```

生成的目标程序较短



6.4.2 defined运算符

- **defined**是预处理运算符，其形式为：

defined（标识符）或defined 标识符

它用来判断标识符是否被**#define**定义了，如被定义，值为**1**，否则为**0**

- **#ifdef R**

同

#if defined(R)

6.4.3 条件编译的应用

【例6.5】采用条件编译，避免多次包含同一个头文件。

- 为了避免一个头文件被多次包含，可以在头文件的最前面两行和最后一行加上预编译指令，让头文件在被多个源文件引用时不会多次编译。

```
#ifndef _NAME_H
#define _NAME_H          /* 定义头文件的标识符 */
.....                  /* 头文件的内容 */
#endif
```

其中，**NAME**是头文件的名字。比如头文件为**myFile.h**，则其标识符可为**_MYFILE_H**。

- 在创建一个头文件时，用**#define**指令为它定义一个唯一的标识符。通过**#ifndef**指令检查这个标识符是否已被定义，如果已被定义，则说明该头文件已经被包含了，就不要再次包含该头文件，**#ifndef**就帮助编译器跳过直到**#endif**的所有文本；反之，则定义这个标识符，以避免以后再次包含该头文件。



调试程序时临时忽略一些代码

- 可以把代码放在注释中。然而，如果代码中也含有注释，就会导致语法错误。
- 使用条件编译能解决这个问题

#if 0

不编译的代码

#endif

- 要让编译器编译这段代码，把原来的**0**改为**1**就可以了。



调试程序时跟踪程序的执行

- 为调试而增加的语句放在条件编译指令之间，在调试时编译这些语句。如：

```
#ifdef DEBUG
```

```
printf("Variable x=%d\n",x);
```

```
#endif
```

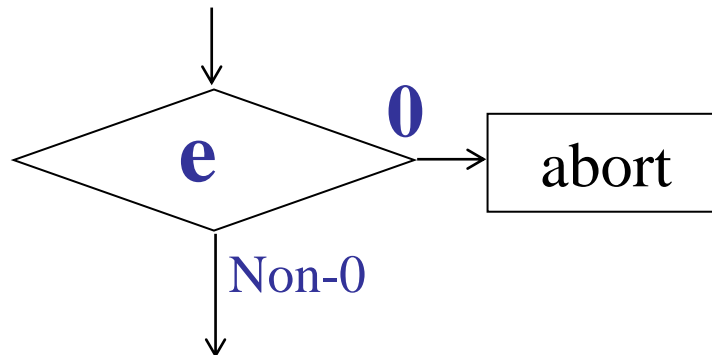
- 在调试程序时，在前面加**#define DEBUG**，就编译**printf**语句，输出供判断参考的**x**值。完成调试后，从程序中去掉**#define**指令，编译就忽略为调试而插入的**printf**语句，相当于它被“自动”删除了。

6.5 assert宏(C中的断言)

在头文件**assert.h**中，用来测试表达式的值是否符合要求，其形式如下：

assert(e)

如果**e**值非0，程序继续执行下一个语句。如果**e**值0，就输出错误信息，并通过调用实用库中的函数**abort**终止程序的执行。



断言与防御性编程

防御性编程使用户在运行程序（发布版本里）时，当出现意外情况时程序仍能继续工作。

```
long Fact(int n)↵
{↵
    int i;↵
    long result = 1;↵
    assert(n >= 0); /* 使用断言检查入口参数的合法性 */↵
    for (i=2; i<=n; i++) ↵
        result *= i;↵
    return result;    ↵
}↵
```

如果 $n < 0$ ，会输出包含行号和文件名的错误信息并中断执行：

Assertion failed: $n \geq 0$, file test.c, line 32

把断言看作一种简单的制造栅栏的方法，这种栅栏能使错误在穿过自己时暴露。