C语言程序设计
The C Programming Language

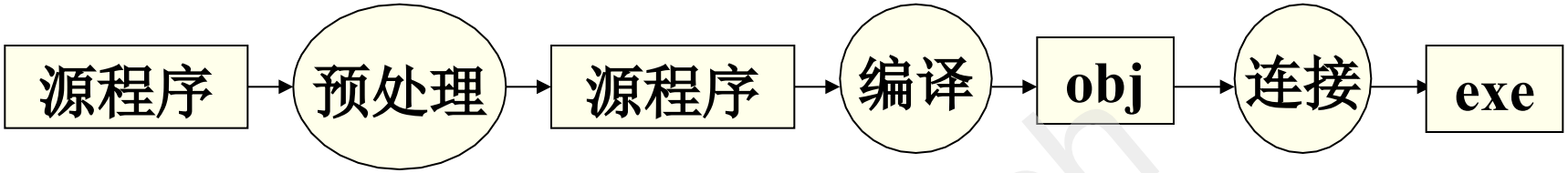
第6章　编译预处理

武汉光电国家研究中心
李春花

本章重点

- 宏定义#define
  - 无参宏定义
  - 带参宏定义 √
- 条件编译

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，

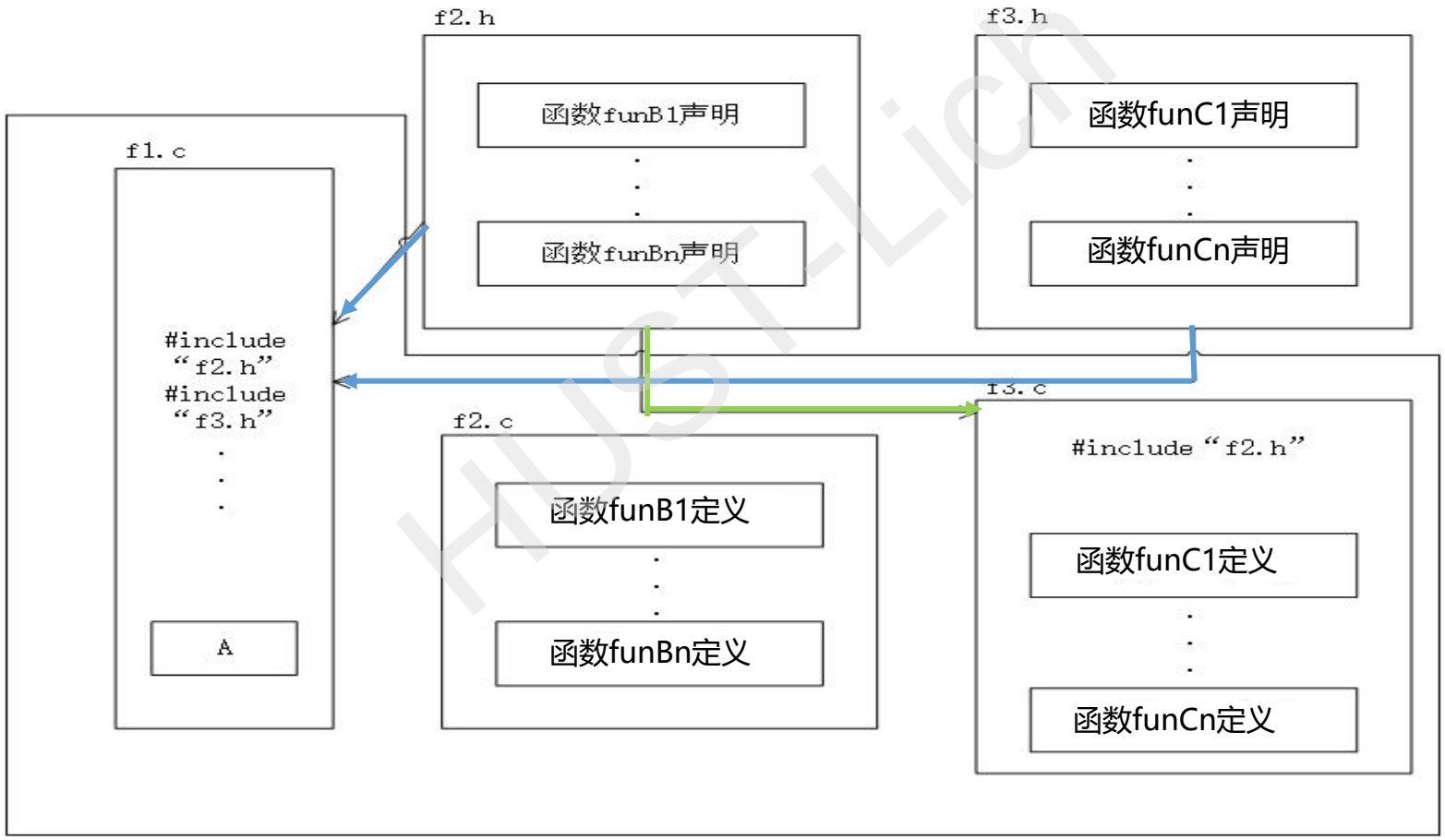
若找不到，再在指定的标准目录下寻找

```
#include <d:\test\myfile.h>
#include “d:\test\myfile.h"
```

采用绝对路径，两者没区别

文件包含

文件包含命令功能：把指定的文件插入该命令行位置取代该命令行，从而把指定的文件和当前的源文件连成一个源文件。



6.2 宏定义#define

用一个标识符来表示一个字符串.

```
#define TRUE    1
#define FLASE   0
```

#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

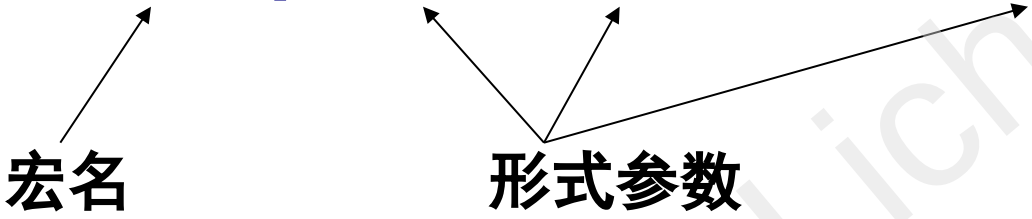
```
#define   M   (y*y+3*y )
void main(void)
{  int  s,y;
   printf("Input a number: ");
   scanf("%d",&y);
   s=3*M+4*M+y*M;
   printf("s=%d\n",s);
}
```

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);
}
```

带参数的宏定义

#define 标识符(标识符，标识符，…，标识符) 字符串



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

#define MAX(x，y) ((x)>(y)?(x):(y))

x^2 的宏定义

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

哪个正确呢？

```
#define  MAX(x,  y)   x>y ? x:y
#define  MAX(x,  y)   (x)>(y)?(x):(y)
#define  MAX(x,  y)   ((x)>(y)?(x):(y))
```

为什么要这么多的括号

```
#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 */

```
#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 */

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

宏调用： 27/SQ(3)

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

注意

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。

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*/

宏的特点

- 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。

取消宏定义#undef

终止宏名的作用域，形式为：

#undef 标识符　　如 #undef PI

何时使用#undef指令?

- 防止宏名的冲突

```
#include "everything.h"
#undef  SIZE      /*若everything.h中定义了SIZE，就取消它；
                    否则后面的同名指令不起作用*/
#define  SIZE  100
```

- 保证调用的是一个实际函数而不是宏

```
#undef  getchar
int  getchar(void) {…}
```

6.3 条件编译

- 条件编译：只对源程序中满足条件的部分进行编译
- 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，即**按照不同的条件去编译程序的不同部分**，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。
- 条件编译指令三种形式，控制流与if语句类似。

  #if　#ifdef　#ifndef

6.3 条件编译

- 条件编译指令三种形式，控制流与if语句类似。

#if　#ifdef　#ifndef

#if 常量表达式
　程序段
#elif 常量表达式
…
#else
　程序段
#endif

#ifdef 标识符
　程序段
#elif 常量表达式
…
#else
　程序段
#endif

#ifndef 标识符
　程序段
#elif 常量表达式
…
#else
　程序段
#endif

利用R计算圆或正方形的面积

预处理前源码

```
#define R 1  // #define R
int main(void)
{
    float r, s;
    printf ("input a number: ");
    scanf("%f ", &r);
    #if R    // #ifdef R
      s=3.14159*r*r;
      printf("%f\n", s);
    #else    // #elif
      s=r*r;
      printf("%f\n", s);
    #endif
    return 0;
}
```

预处理后源码

```
int main(void)
{
    float r, s;
    printf ("input a number: ")
    ;
    scanf("%f ", &r);
    s=3.14159*r*r;
    printf("%f\n", s);
    return 0;
}
```

生成的目标程序较短

利用R计算圆或正方形的面积

预处理前源码

```
#define  R  0  // #define R
void  main(void)
{
    float  c，  s;
    printf ("input a number: ");
    scanf("%f ",  &c);
    #if  R     // #ifdef R
       s=3.14159*c*c;
       printf("%f\n",  s);
    #else
       s=c*c;
       printf("%f\n",  s);
    #endif
}
```

预处理后源码

```
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

条件编译的应用--调试程序

- 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

```
#define DEBUG  //完成调试后，去掉该#define指令即可
...
#ifdef  DEBUG
    printf("x=%d\n",x);
#endif
...
#ifdef  DEBUG
    printf("y=%d\n",y);
#endif
```

条件编译的应用—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处理指令，让头文件被多个源文件引用时不会被多次编译

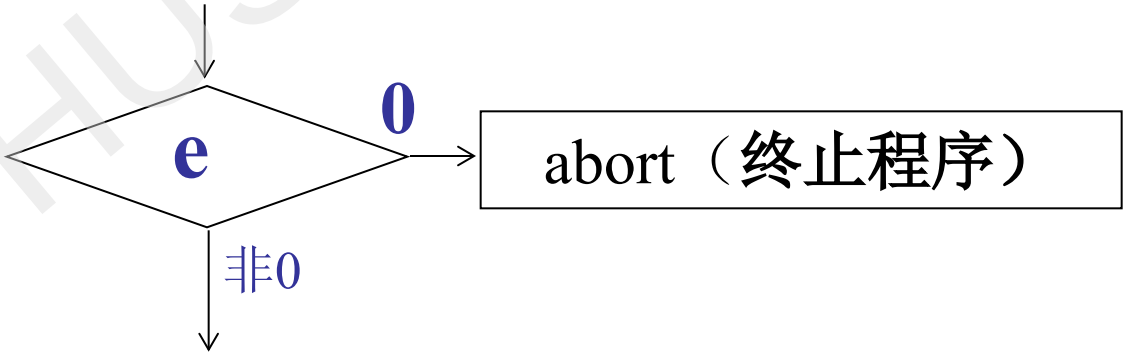
```
#ifndef _MYFILE_H
  #define _MYFILE_H   //定义头文件标识符
   ...                //头文件的内容
#endif
```

断言--assert宏

- 在头文件assert.h中
- 测试表达式的值是否符合要求

  assert(e)

- 执行流程



assert宏

```
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>=0, file test.c,line 32