西北工業大學

NORTHWESTERN POLYTECHNICAL UNIVERSITY

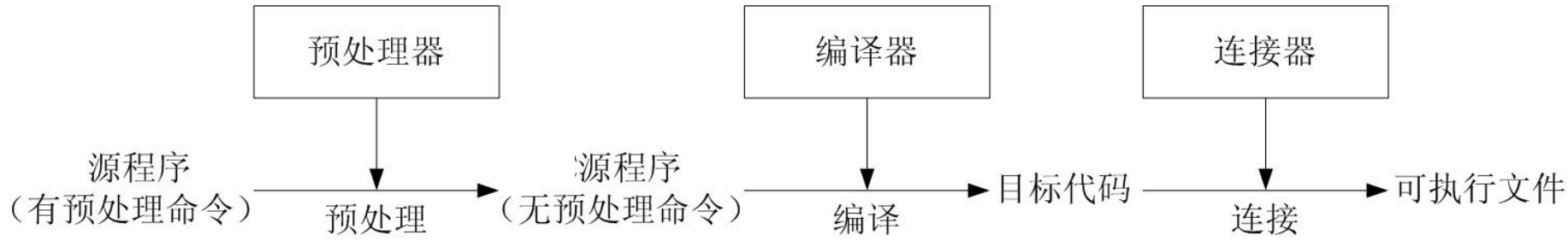
程序设计基础
Programming in C++

U10G13027/U10G13015

主讲：魏英，计算机学院

- 5.1 宏定义
- 5.2 文件包含
- 5.3 条件编译
- 5.4 其他命令

图5.1 编译、连接处理过程



预处理命令不是C++本身的组成部分，更不是C++语句，它是C++标准规定的可以出现在C++源程序文件中的命令。

这些命令必须以“#”开头，结尾不加分号，可以放置在源程序中的任何位置，其有效范围是从出现位置开始到源程序文件末尾。

- 在C++源程序中允许用一个标识符来代表一个字符文本，称为宏，标识符为宏名。

- 宏定义通常用于定义程序中的符号常量、类型别名、运算式代换、语句代换等，其命令为#define，分为不带参数的宏定义和带参数的宏定义。

- 不带参数的宏定义
- 不带参数的宏定义的命令形式为：

```
#define   宏名   字符文本
```

```
#define PI   3.1415926
L=2*PI*r;

#define M   y*y+5*y
S=3*M+4*M+5*M;
```

```
L=2*3.1415926*r;

S=3*y*y+5*y+4*y*y+5*y+5*y*y
+5*y;
```

- 宏定义只是简单置换，不作语法检查，因此，宏串中的每字符都是有效字符；
- 下例多余字符均导致编译出错：

```
#define  PI  "3.141592"
c=2.0*PI*r;     宏展开结果：c=2.0*"3.141592"*r;

#define  PI  3.141592;
c=2.0*PI*r;     宏展开结果：c=2.0*3.141592;*r;
```

- 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)
```

- 例如有宏定义

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

- 程序代码：

```
L=max(x-y,x+y); //max宏引用
```

- 预处理时宏替换为：

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

例5.2

```
1  #include <iostream>
2  using namespace std;
3  int M1(int y)
4  {
5      return((y)*(y));
6  }
7  #define M2(y) ((y)*(y))
```

例5.2

```
 8  int main()
 9  {
10      int i,j;
11      for (i=1,j=1;i<=5;i++)
          cout<<M1(j++)<<" "; //函数调用处理
12      cout << endl;
13      for (i=1,j=1;i<=5;i++)
          cout<<M2(j++)<<" "; //宏引用处理
14      cout << endl;
15      return 0;
16  }
```

- 为了保证宏展开的结果符合设计本意，应在宏串或实参字串中加入必要的括号；

例：计算以a+b为半径的圆面积。

```
#define S(r)  3.141592*(r)*(r)
int main( )
{ float a=2.0, b=3.0, area;
  area= S(a+b)
  cout<<"area="<<area<<endl;
  return 0;
}
```

- #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++字符串常量。

```
#define PRINT_MSG1(x) printf(#x);
#define PRINT_MSG2(x) printf(x);
PRINT_MSG1(Hello World); //正确
PRINT_MSG1("Hello World"); //正确
PRINT_MSG2(Hello World); //错误
PRINT_MSG2("Hello World"); //正确
```

- ##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。

```
#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;
```

- 文件包含
- 文件包含命令的作用是把指定的文件插入到该命令所处的位置上取代该命令，然后再进行编译处理，相当于将文件的内容“嵌入”到当前的源文件中一起编译。

- 文件包含
- 文件包含命令为#include，有两种命令形式：
- ①第一种形式：

```
#include <头文件名>
```

- ②第二种形式：

```
#include "头文件名"
```

- 1. 文件包含的路径问题
- 文件包含命令中的头文件名可以写成绝对路径的形式，例如：

```
#include "C:\DEV\GSL\include\gsl_linalg.h"
#include <C:\DEV\SDL\include\SDL.h>
```

- 头文件名也可以写成相对路径的形式，例如：

```
#include <math.h>
#include <zlib\zlib.h>
#include "user.h"
#include "share\a.h"
```

- 这时的文件包含命令是相对系统INCLUDE路径或用户路径来查找头文件的。
- 假设编译器系统INCLUDE路径为“C:\DEV\MinGW\include”，则

- 假设用户路径为“D:\Devshop”，则

```
#include "user.h"
   //user.h在D:\Devshop或C:\DEV\MinGW\include
#include "share\a.h"
   //a.h 在D:\Devshop\share
         或C:\DEV\MinGW\include\share
```

- 如果在上述路径中找不到头文件，会出现编译错误。

- 2．文件包含的重复包含问题
- 头文件有时需要避免重复包含（即多次包含），例如一些特定声明不能多次声明，而且重复包含增加了编译时间。这时可以采用以下两个办法之一。
- （1）使用条件编译。例如：
- （2）使用特殊预处理命令#pragma，例如：
- #pragma once

- 使用条件编译，可以针对不同硬件平台和软件开发环境来控制不同的代码段被编译，从而方便了程序的可维护性和可移植性，同时提高了程序的通用性。

- #define定义条件
- 条件编译使用宏定义条件，其命令形式为：

```
#define   条件字段
#define   条件字段  常量表达式
```

```
#define DEBUG
#define WINVER 0x0501
```

- #ifdef条件编译命令测试条件字段是否定义，以此选择参与编译的程序代码段，它有两种命令形式。
- ①第一种形式：

```
#ifdef   条件字段
      ......//程序代码段1
#endif
```

- ②第二种形式：

```
#ifdef   条件字段
       ......//程序代码段1
#else
        ......//程序代码段2
#endif
```

- 表示如果DEBUG已经定义则编译printf语句，否则不编译；

```
#ifdef DEBUG
printf("x=%d, y=%d, z=%d\n", x, y, z);
#endif
```

- 无论if语句条件满足与否，程序可执行代码中是肯定有printf语句指令的，if语句条件用来决定是否执行它。

```
if (DEBUG)
    printf("x=%d, y=%d, z=%d\n", x, y, z);
```

- #if条件编译命令根据表达式的值选择参与编译的程序代码，其命令形式为：

```
#if   常量表达式
        ......//程序代码段1
#else
        ......//程序代码段2
#endif
```

- 可以使用嵌套的#if条件编译命令#if-#elif，命令形式为：

```
#if   常量表达式1
        ......//程序代码段1
#elseif   常量表达式2
        ......//程序代码段2
#else
        ......//程序代码段3
#endif
```


程序设计