

C++高级语言程序设计

• 王晨宇

• 北京邮电大学网络空间安全学院

第3章 编译预处理和程序的多文件组织

3.1 宏定义

3.2 文件包含

3.3 条件编译

3.4 程序的多

文件组织

- **编译预处理命令**：以#开头，以回车结束，独占一行；可出现在程序中的任何位置，常置于源程序的开始；不属于C++的语法范畴。
- **常用编译预处理命令**：宏、文件包含和条件编译。
- **编译预处理**：编译源程序前，先用预处理程序处理源程序中的编译预处理命令，并过滤源程序中的注释和多余空白符，生成一个完全用C++语言表达的临时源程序文件供编译系统处理。

3.1 宏定义

- 宏定义：用预处理命令#define实现。
- 宏定义分为：
 - (1)带参数的宏定义；
 - (2)不带参数的宏定义。

3.1.1 不带参数的宏定义

- 格式:

`#define` 标识符 字符或字符串

其中，标识符称为宏名。

- 举例:

`#define PI 3.1415926`

其作用是将宏名PI定义为字符串3.1415926。在编译预处理时，将该命令后所有出现PI处均用3.1415926替换。这种替换称为“宏替换”或“宏扩展”或“宏展开”。

- 宏替换的本质是字符串替换。

- 又如:

`#define PROMPT "面积为:"`

- 例3.2 宏定义的使用。

```
#include<iostream>
using namespace std;
#define PI 3.1415926
#define R 2.8
#define AREA PI*R*R          //A
#define PROMPT "面积 = "
int main(void)
{ cout<<PROMPT<<AREA<<"\n" ;
  return 0;
}
```

- 程序运行结果:
面积 = 24.6301

不带参宏说明

- (1)通常宏名用**大写字母**表示，以别于变量名。当然，从语法上来讲，任一合法的标识符均可用作宏名。
- (2)宏定义可出现在程序中的任何位置，但通常放在源程序**文件的开始**。宏名的作用域从宏定义开始到本源程序文件结束。
- (3)在宏定义中可用已定义的宏名。

如例3.2中的A行，在宏定义AREA时，用到已定义的宏名PI和R。在编译预处理时，先对该行中的PI和R作替换。替换后，A行为：

```
#define AREA 3.1415926*2.8*2.8
```

(4)宏扩展时只对宏名作字符串代换，不作任何计算，也不作任何语法检查。若宏定义时书写不正确，会得到不正确的结果或编译时出现语法错误。如：

```
#define A 3+5
```

```
#define B A*A
```

```
cout<<B<<'\n';    //C
```

C行输出为23，而不是64。因C行宏扩展后为：

```
cout<<3+5*3+5<<'\n';
```

(5)若要终止宏名的作用域，可用预处理命令：

```
#undef 宏名
```

例如：

```
#define PI 3.1415926
```

```
...
```

```
#undef PI //终止PI的作用域，其后不能再PI
```

(6)当宏名出现在字符串中时，编译预处理不做宏扩展。

例3.3宏名出现在字符串中时的编译预处理。

```
#include<iostream.h>
#define A "欢迎"
#define B "A参观展览会"
void main(void)
{ cout<<B<<"! \n"; }
```

程序运行结果：

A参观展览会!

(7)在同一个作用域内，同一个宏名不允许重复定义。

3.1.2 带参数的宏定义

- 带参数宏定义的形式:

`#define` 宏名(参数表) 使用参数的字符或字符串
带参宏定义在宏扩展时, 要对宏名和参数做替换。

- 带参数宏举例:

```
#define AREA(a,b) a*b  
b=AREA(2.0,7.8);
```

宏调用: 使用带参宏。在宏调用中给出的参数称为实参。

- 宏定义中的参数称为形参。
- 例如: 宏AREA有参数a和b。

- 带参宏扩展: 先用实参替代宏定义中的形参, 并将替代后的字符串替代宏调用。宏扩展仅作字符串替代, 不作计算。
- 举例: 宏调用经宏扩展后为:
b=2.0*7.8;

带参宏说明

(1)宏调用中的实参若含表达式，则在宏定义中要用圆括号把形参括起来或在宏调用中把实参括起来，以免出错。例如：

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

```
c=AREA(2+3,3+4);           //B
```

c的值不为35，而是15。

- 出错原因：因B行扩展后为：

```
c=2+3*3+4;
```

- 解决方法：

①将B行的宏调用改写成：

```
c=AREA((2+3),(3+4));
```

②将宏定义改为：

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

则B行经宏扩展后，成为：

```
c=(2+3)*(3+4);
```

方法②更好。

(2)在宏定义时，宏名与左括号之间**不能有空格**。若在宏名后有空格，则将空格后的所有字符都作为无参宏所定义的字符串，而不作为形参。例如：

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

则编译预处理程序认为无参宏AREA定义为“(a,b)(a)*(b)”，而不将(a,b)作为参数。

(3)当一个宏定义多于一行时必须使用续行符“\”，即在按换行符(Enter键)之前先输入一个“\”。例如：

```
#define AREA(a,b) (a)*\  
                (b)
```

(4)带参宏主要用来取代功能简单、代码短小、运行时间极短、调用频繁的程序代码，但因其使用时有一些副作用，故C++引入内联函数以取代带参宏。

带参宏与函数有些相似，但两者有本质区别

- (1)定义形式不同。宏定义只给出形参，而不指明形参类型；而函数定义必须指定每个形参的类型。
- (2)调用处理不同。宏由编译预处理程序处理，而函数由编译程序处理。宏调用仅作替换，不做任何计算；而函数调用是在目标程序执行期间，先依次求出各个实参的值，然后才执行函数的调用。
- (3)函数调用要求实参类型必须与对应的形参类型一致，即做类型检查；而宏调用没有参数类型检查。
- (4)函数可用return语句返回一个值，而宏不返回值。
- (5)多次调用同一个宏时，要增加源程序的长度；而对同一个函数的多次调用，不会使源程序变长。

3.2 “包含文件” 处理

- 文件包含：一个源文件可将另一个源文件的全部内容包含进来。通过编译预处理命令include实现。
- 文件包含命令：

格式1： `#include<filename>`

按编译系统规定的路径(include目录或其子目录)查找包含文件。主要用于包含编译系统预定义的头文件。

格式2： `#include"filename"`

从当前目录查找包含的文件。主要用于包含用户自定义的文件。

文件包含举例

- 设文件my.h的内容为:

```
int x=2;
```

- 设文件f.cpp的内容为:

```
#include "my.h"
```

```
void main(void){ cout<<x<<"\n"; }
```

- 上述程序经过编译预处理后，用文件my.h的内容替换编译预处理命令行产生一个临时文件，其内容为:

```
int x=2;
```

```
void main(void){ cout<<x<<"\n"; }
```

并把该临时文件交给编译程序进行编译。

文件包含说明

- (1)包含文件的扩展名常用“.h”(head的缩写)。当然也可用其他扩展名或不用扩展名。
- (2)一个include命令只能指定一个被包含的文件，若要包含n个文件，则要用n个include命令。
- (3)在一个包含文件中可包含其他包含文件。
- (4)include命令可出现在程序中的任何位置，通常放在程序的开头。
- (5)为了便于文件包含，允许在文件名前加路径，例如：
`#include "c:\\my\\my.h"`

头文件

- 头文件：因包含文件的扩展名 “.h” (head) 而得名。
- 头文件的内容：声明公用的数据类型、函数原型、宏定义、全局变量、命名空间等。
- 头文件的作用与用途：
 - (1)头文件是后续编程的约定；
 - (2)头文件便于后续编程时引用，便于时刻检查引用头文件的源程序是否严格遵守编程约定，为程序的模块级编程与调试、系统级编程与调试带来诸多便利。
 - (3)头文件是实现代码重用的好办法之一。
 - (4)头文件为设计复杂的或多人合作的程序提供了便利。

函数原型声明在头文件中

```
//month.h头文件
void printMonth(int year, int month);
void printMonthTitle(int year, int month);
void printMonthBody(int year, int month);
int getStartDay(int year, int month);
int getTotalNumberOfDays(int year, int month);
int getNumberOfDaysInMonth(int year, int month);
bool isLeapYear(int year);
```

函数的实现代码

```
//month.cpp
#include <iostream>
#include <iomanip>
#include "month.h"
using namespace std;
.....
```

使用时要在cpp文件的最开始使用
include将要用的头文件包含进来。

主程序中调用其它函数

```
//mainprog.cpp主函数的实现代码
#include <iostream>
#include <iomanip>
#include "month.h"
using namespace std;

void main()
{ cout << "请输入年份 (如2010) : ";
  int year;
  cin >> year;
  cout << "请输入月份 (1 - 12) ";
  int month;
  cin >> month;
  printMonth(year, month); //打印月历
}

void printMonth(int year, int month) //打印一年中某月的月历
{ printMonthTitle(year, month); //打印月历头部
  printMonthBody(year, month); //打印月历主体
}
.....
```

3.3 条件编译

- 条件编译：程序中的某些行仅在满足某种条件时，才要编译程序对其编译；否则不被编译。
- 条件编译命令分类：
 - (1)传统条件编译命令：根据宏名是否已经定义来确定是否要编译某些程序行；
 - (2)现代的条件编译命令：根据表达式的值来确定是否要编译某些程序行。
- 条件编译命令可出现在程序中的任何位置。编译预处理程序在处理条件编译时，将要编译的程序段写到一个临时文件中，供编译程序编译。

传统条件编译命令

格式一 #ifdef 宏名
 程序段

#endif

格式二 #ifdef 宏名
 程序段 1

#else

程序段 2

#endif

格式三 #ifndef 宏名
 程序段

#endif

格式四 #ifndef 宏名
 程序段 1

#else

程序段 2

#endif

现代的条件编译命令

格式五 `#if` 常量表达式
 程序段

`#endif`

格式六 `#if` 常量表达式
 程序段 1

`#else`

程序段 2

`#endif`

格式七 `#if` 常量表达式 1
 程序段 1

`#elif` 常量表达式 2
程序段 2

`#elif` 常量表达式 3
...

`#else`

程序段 n

`#endif`

条件编译用于程序调试

- 程序调试时常要输出调试信息，而调试结束不输出，则可将输出调试信息的语句用条件编译括起来，如：

```
#ifdef DEBUG  
    cout<<"x="<<x<<"\n";  
#endif
```

调试程序期间，在源程序的开头增加宏定义：

```
#define DEBUG
```

实现了输出调试信息的目的。一旦程序调试结束，只要删除DEBUG的宏定义，重新编译程序，则所有的输出调试信息的程序部分均不编译。

条件编译防止包含文件重复包含

设文件a2.h的内容:

```
float area;
```

设文件a1.h的内容:

```
#include "a2.h"
```

```
#define PI 3.1415926f
```

```
#define R 2.8f
```

编译源程序文件a.cpp时出错。原因是变量area重复定义。

源程序文件a.cpp的内容:

```
#include <iostream>
```

```
using namespace std;
```

```
#include "a1.h"
```

```
#include "a2.h"
```

```
int main(void)
```

```
{ area=PI*R*R;
```

```
cout<<"圆面积="
```

```
<<area<<"\n" ;
```

```
return 0;
```

```
}
```

- 解决办法:

(1)同一头文件在源程序文件中保证只包含一次。简单程序可以做到，但复杂程序很难做到。

(2)定义头文件时，使用条件编译，以保证同一头文件不论被包含多少次，只有第一次的包含命令起作用。例如，上面的头文件a2.h改写为：

```
#ifndef _A2_H          //A
    #define _A2_H      //B
    float area;
#endif                //C
```

条件编译命令将头文件的内容括起来。

3.4 程序的多文件组织

- **单文件组织程序：** 一个程序放在一个源程序文件中。
- **多文件组织程序：** 一个程序放在一个以上的文件中。
- **多文件组织程序的优势：** 便于代码的设计、调试以及重用；便于多人协同编写同一个程序等。
- **多文件组织程序涉及：** 编译、链接，一个文件中的函数调用另一个文件中的函数或使用另一个文件中的全局变量等。

3.4.1 内部函数和外部函数

- **内部函数也称静态函数**：仅限于本文件内使用的函数。在定义函数时，函数的类型前加static，如：

```
static float f( )//内部函数  
{ ... }
```

- **外部函数也称全局函数**：既可在本文件内使用，也可在其他文件中使用的函数。在**定义函数**时，函数的类型前加extern或省略extern，如：

```
extern int f1( )//外部函数：用extern修饰  
{ ... }  
float f2( )    //外部函数：省略extern  
{ ... }
```

外部函数应用举例

- 文件c1.cpp定义函数f():

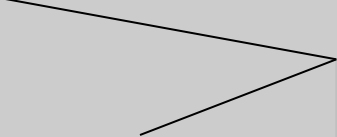
```
int f(int x)
{ return x+1; }
```

- 在文件c2.cpp中调用文件c1.cpp中定义的函数f(), 在调用前必须增加如下函数原型说明:

```
extern int f(int);
```

```
...
```

```
int i=f(x);
```



函数原型说明,将f()函数的作用域扩展到本文件, 以便此后调用。

外部变量

- 文件t1.cpp的内容为:

```
extern int x=2;
```

- 文件t2.cpp的内容为:

```
#include<iostream.h>
void main(void)
{ extern int x;
  cout<<x<<"\n";
}
```

上述两个程序文件经编译链接, 并运行后, 输出:

x=2

- 全局变量x的定义性说明(extern可省), 一是获得了静态存储区的内存和初值, 二是允许本文件和其它文件使用。

- 若改成:

```
extern int x;
```

则是引用性说明。编译上述两个程序文件时无错, 但链接时出错, 原因是找不到全局变量x。

t2.cpp文件的main()函数对外部变量x做了引用性说明, 扩展了外部变量x的作用域, 使main函数可以访问外部变量x。

多文件组织程序的原则

- 将一个程序按功能(面向过程)或按类(面向对象)分成若干模块，再将每个模块程序的接口和实现分离，分存于指定的头文件(".h")和实现文件(".cpp")中。
- 头文件是模块的接口，提取的是模块实现(对内)和使用(对外)的关键信息，通常包含模块中的全局类型定义、函数原型声明、全局常量和变量的定义、模板和命名空间的定义、编译预处理命令、注释等。
- 实现文件是模块的实现细节。首先，在文件开始，用包含命令包含本模块的头文件；其次包含模块中的函数定义、类类型的成员函数定义、编译预处理命令、注释等。

面向过程的多文件组织程序实例

- 例3.4 将例1.2的源程序改写成多文件程序。程序由3个文件组成：头文件Circle.h是Circle模块的接口，声明了函数area的原型；实现文件Circle.cpp是Circle模块的实现细节，即函数area的定义；程序的主文件mymain.cpp是Circle模块的使用。

(1)Circle.h

```
#ifndef _CIRCLE_H
#define _CIRCLE_H
const float pi=3.1415f;
float area(float r);
#endif
```

(2)Circle.cpp

```
#include "Circle.h"
float area(float r)
{ return pi*r*r; }
```

(3)mymain.cpp

```
#include <iostream>
using namespace std;
#include "Circle.h"
void main(void)
{ float r;
  cout<<"输入圆的半径:";
  cin>>r;
  cout<<"半径为"<<r
    <<"的圆的面积="
    <<area(r)<<"\n";
}
```

面向过程的多文件组织程序实例

- 头文件Circle.h中，应有避免其本身被重复包含的条件编译命令。
- 在模块Circle的实现文件Circle.cpp和使用模块Circle的主文件mymain.cpp中，都包含头文件Circle.h。前者，用Circle模块的接口来约束Circle模块的实现。这样，模块的实现中若违反接口约定，则会在编译时报错。后者，通过Circle模块的接口来规范Circle模块的使用。一旦违规使用Circle模块，也会在编译时报错。由此可见，头文件Circle.h作为Circle模块的接口，对于Circle模块的实现和使用都至关重要。

3.4.2 多文件组织程序的编译和链接

- 编译、链接一个由多文件组成的程序，有下列方式：
 - (1)包含命令方式；
 - (2)单独编译方式；
 - (3)工程(或项目)文件方式。

包含命令方式

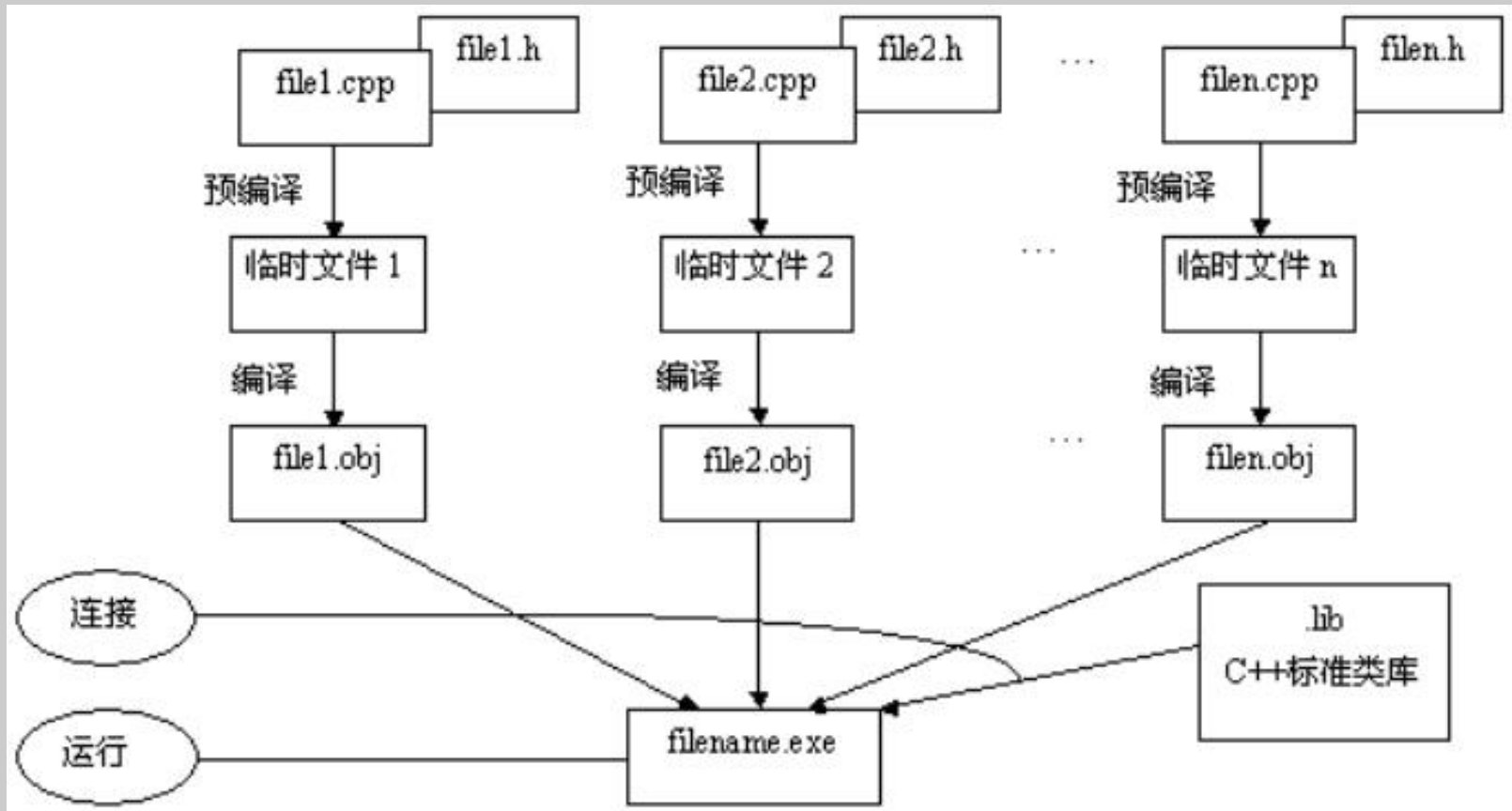
- 在定义main()函数的文件中将组成同一程序的其他文件用包含命令包含进来，由编译程序对这些源程序文件一起编译，并链接成一个可执行的文件。这种方法比较适合规模不大的程序。但对于大程序来说，这种方法有明显不足：
 - (1)对任一文件的微小修改，均要重新编译所有的文件，然后才能链接，非常费时；
 - (2)用包含文件的方式将所有文件都包含在一个临时文件中，使所有的全局性标识符(如全局变量、全局类型、函数等)都在同一个文件作用域内，使得全局性标识符的重名问题、干扰问题非常突出，非常棘手。

单独编译

- 将各个源程序文件单独编译成目标程序，然后用操作系统或编译器提供的链接程序将这些目标程序文件链接成一个可以执行的程序文件。
- 这是最原始的多文件组织方法，通常在命令行状态下操作，需要记忆常用的编译选项。虽然，目前的C++编译器均支持这种方式，但很少有非专业人士愿意使用。

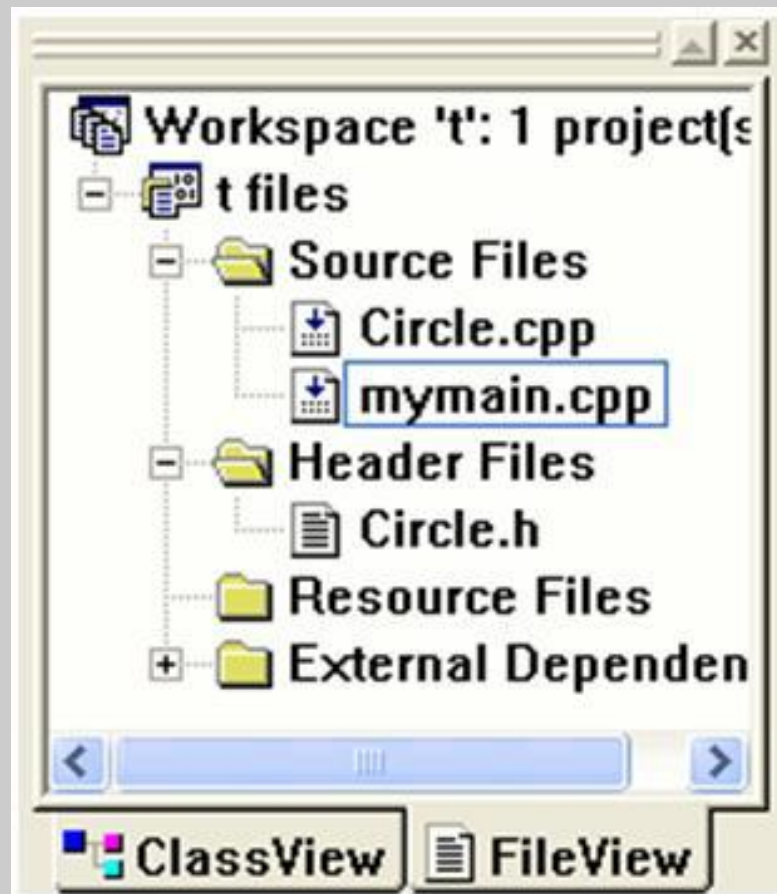
工程(或项目)文件

- 编译时每个源程序文件单独编译，如果源程序文件中有编译预处理指令，则首先经过编译预处理生成临时文件存放在内存，之后对临时文件进行编译生成目标文件.obj，编译后临时文件撤销。所有的目标文件经连接器连接最终生成一个完整的可执行文件.exe。



工程(或项目)文件

- 将组成一个程序的所有文件都加到工程(或项目)文件中，由编译器自动完成这些文件的编译和链接。这是多数流行C++ IDE推荐的方法，使用简便。
- 有关工程(或项目)文件的详细说明，可查阅有关C++编译器的使用手册。
- 右图给出了例3.4在VC++6.0 IDE中的项目文件结构。



(4) 作业

用多文件方式实现各科成绩的输入，平均分、最高分、最低分计算，要求用宏实现其中的部分函数，并用到条件编译