高级语言程序设计

第09章 编译预处理与多文件工程程序

内容提要

大型程序开发：

- 编译预处理
- 组织多文件工程程序
- 模块化程序设计

9.1 编译预处理

- 编译预处理（Preprocessor）就是编译器根据源程序中的编译预处理指令对源程序文本进行相应操作的过程。

- 编译预处理的结果是一个删除了预处理指令、仅包含C语言语句的新的源文件，该文件才被正式编译成目标代码。

⇩

完成该过程的指令就是预处理命令

9.1 编译预处理

南京邮电大学 Nanjing University of Posts and Telecommunications

- C语言的编译预处理指令主要包括三种：

文件包含（Including files） → 宏定义（Macro Definition） → 条件编译（Conditional Compilation）

➢ 功能分别用宏定义命令、头文件包含命令、条件编译命令来实现

9.1 编译预处理

- 编译预处理指令以”#”开头，结尾不加”;”号

```
# define PI 3.1415926
# include <stdio.h>
```

在程序中，我们看到以#开头的都是编译预处理指令，因为它们不是C语言语句，所以不用分号结束。

文件包含

- 头文件（Header File） 什么作用？
  - 主要的作用是保存程序的声明，包括：功能函数原型、数据类型的声明等，如：stdio.h、math.h、string.h等。
- 文件包含指令的一般格式为：
  - #include <头文件名>
    - 到编译系统指定的标准目录（\include）下去查找该头文件，若没有找到就报错。多用于包含标准头文件。
  - #include "头文件名"
    - 首先到当前工作目录中查找头文件；若没找到，再到查找编译系统指定的标准目录中查找。多用于包含用户自定义的头文件。

宏定义

- 宏定义的作用：将一个标识符定义为一个字符串，在编译预处理时，源程序中的该标识符均以指定的字符串来代替。

格式：#define <标识符> <字符串>
例如：# define PI 3.1415926

#define是宏定义命令
标识符称为“宏名”
在预编译时将宏名替换成字符串的过程称为“宏展开”

宏定义

- 无参宏指令
  - #define <标识符> <字符串>
- 例9.1无参宏指令应用示例

```
#define PI 3.14159        /*无参宏定义1，符号常量*/
#define ISPOSITIVE >0                  /*无参宏定义2*/
#define FORMAT  "Area=%f\n"            /*无参宏定义3*/
#define ERRMSG  "Input error!\n"       /*无参宏定义4*/
...
```

宏定义

(1) 宏名一般习惯用大写字母表示，以便与变量名相区别。

(2) 宏定义是用宏名代替一个字符串，只作简单置换，不作正确性检查。只有在编译已被宏展开后的源程序时才会发现语法错误并报错。

(3) #define命令写在文件开头，函数之前，作为文件一部分，在此文件范围内有效。

宏定义

(4) 可以用#undef命令终止宏定义的作用域。这样可以灵活控制宏定义的作用范围。

```
#define G  9.8
 int main()
 {
       ...
 }
#undef G
f1()
{
     ...
 }
```

G的有效范围

在f1函数中，G不再代表9.8。

宏定义

- 带参宏指令

  #define 〈标识符〉（〈参数列表〉） 〈字符串〉

  - 作用：完成一些简单函数所能实现的功能，同时又减少系统开销

  与无参宏定义类似，在带参宏定义中的参数传递也仅仅是一个符号的置换过程，与普通函数中实参与形参之间的值传递机制，有着本质区别

宏定义

- 带参宏指令
  - #define <标识符>(<参数列表>)
- 例9.2　带参宏指令应用示例

```
#define SUB(a,b)  a-b              /*带参宏定义*/
…
c=SUB(a,b);                  /*替换为：c=a-b; */
…
c=SUB(3,1+2);    /*替换为：c=3-1+2; 结果不为0*/
```

思考题：如何修改本例宏定义使得SUB(3, 1+2)的结果为0？

宏定义

- 取消宏定义指令
  - #undef <标识符>

- 例如：
  - #undef PI　　表示取消标识符PI的宏定义

设置

1、C语言编译系统对宏定义的处理_____。

A 和其它C语句同时进行

B 在对C程序语句正式编译之前处理

C 在程序执行时进行

D 在程序链接时处理

提交

设置

Nanjing University of Posts and Telecommunications

2、以下对宏替换的叙述，不正确的是_____。

A 宏替换只是字符串的替换

B 宏替换不占用运行时间

C 宏名无类型，其参数也无类型

D 带参宏替换时先求出实参表达式的值，然后代入形参运算求值

宏替换是在预编译时将整个宏的表达式进行替换再进行编译的

提交

设置

3、宏定义#define G 9.8中的宏名G表示_____。

A 一个单精度实数

B 一个双精度实数

C 一个字符串

D 不确定类型的数

提交

设置

4、对于以下宏定义：

#define M 1+2

#define N 2*M+1

执行语句“x=N;”之后，x的值是_____ 。

A 3

x=2*1+2+1

=2+2+1=5

B 5

C 7

D 9

提交

5、对于以下宏定义：

#define M(x) x*x

#define N(x, y) M(x)+M(y)

执行语句z=N(2, 2+3);后，z的值是 _____。

A 29

B 30

C 15

D 语法错误

M(2)+M(2+3)

=2*2+2+3*2+3

=4+2+6+3

=15

提交

条件编译

- 条件编译：对部分内容指定编译的条件，使其只在满足一定条件下才进行编译。

- 条件编译命令有常用两种形式：

```
(1) #ifdef 标识符
        程序段 1
    #else
        程序段 2
    #endif
```

若<标识符>已被定义过，则编译<程序段1>；否则编译<程序段2>

```
(2) #ifndef 标识符
        程序段 1
    #else
        程序段 2
    #endif
```

若<标识符>没有被定义过，则编译<程序段1>；否则编译<程序段2>

- 例9.3　条件编译指令应用示例
  - 用于程序的调试信息的输出

```
#include<stdio.h>
#include<math.h>
#define DEBUG        /*宏定义指令*/
int main()
{
   double a, b, c;
   double s, area;
   scanf("%lf%lf%lf",&a, &b, &c);
 #ifdef DEBUG          /*条件编译指令*/
   printf("DEBUG: a=%f, b=%f, c=%f\n",a,b,c);
#endif
   s=(a+b+c)/2;
#ifdef DEBUG          /*条件编译指令*/
    printf("DEBUG: s=%f\n",s);
#endif
   area=sqrt(s*(s-a)*(s-b)*(s-c));
   printf("Area=%f\n",area);
   return 0;
}
```

输入：3.0　4.0　5.0<回车>
输出：DEBUG：a=3.000000，b=4.000000，
c=5.000000
　　DEBUG：　s=6.000000
　　Area=6.000000

调试完成后，删除#define DEBUG，条件编译中的语句因条件不符不再被编译

输入：3.0　4.0　5.0<回车>
输出：Area=6.000000

小结

- 无参宏与带参宏的定义与使用
- 如何用条件编译帮助我们调试程序，为大型程序设计做技术准备

9.2 多文件工程程序

建立多文件工程程序：

- 组织多文件工程程序
- 进行模块化程序设计

9.2 多文件工程程序

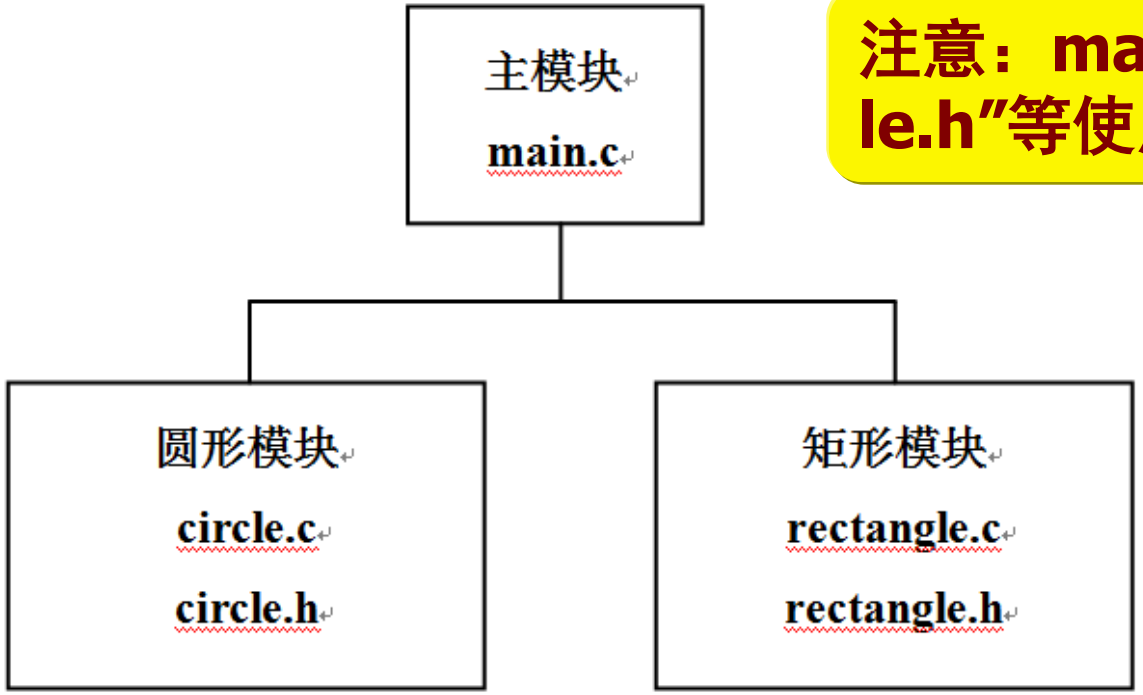
- 多文件工程程序（Project with Multiple Source Files）
  - 程序代码按一定的分类原则被划分为若干个部分，也称为模块（Module）
- 优势：
  - 使程序结构更加清晰
  - 便于程序的分工协作开发
  - 便于程序的维护

多文件工程程序的组织结构

- 将不同的功能和数据结构划分到不同的模块中
  - 不同类型的程序放在不同的（.c）源文件中
- 将函数的定义和使用相分离
  - 将函数的定义从程序其他代码中分离出来，单独存放，有利于函数的重用
- 将函数的声明和实现相分离
  - 将函数的原型声明放在一个（.h）头文件中
  - 将函数的具体实现放在另一个（.c）源文件中

多文件工程程序的组织结构

- 例9.4 设计一个多文件工程程序，其功能是计算圆和矩形的面积和周长。



注意：main.c中#include “circle.h”等使用了双引号

- 在VC++环境下分析程序结构。

小结

- 如何来划分复杂程序的模块
- 如何组织多文件工程程序
- 通过案例学习建立多文件工程程序的过程

外部变量与外部函数

- 外部变量与外部函数
- 静态全局变量与静态函数

外部变量与外部函数

- 在多文件工程程序中，不同文件之间往往需要共享信息。

在一个文件中定义的变量或函数如何能被其他文件所使用呢？

外部变量与外部函数

- 在多文件工程程序中，不同文件之间往往需要共享信息。

在C语言中，我们可以通过外部变量（External Variable）和外部函数（External Function）声明来实现这一目标，具体格式如下：

- 外部变量（External Variable）
  - extern <全局变量定义>;
- 外部函数（External Function）
  - extern <函数声明>;

必须是已定义的全局变量，而不是局部变量。

extern仅是作变量声明，不是定义变量。

外部变量与外部函数

变量的定义 → 需要建立存储空间 → 例如：int a,b,c;

变量的声明 → 不需要建立存储空间 → 例如：extern int a,b;

外部变量的定义只能有一次，其位置在所有函数之外，而在多文件程序中，可以在不同的文件中多次对同一外部变量的声明。这样的声明不分配存储空间，仅仅是为了扩展该变量的作用范围。

外部变量与外部函数

- 例9.5 外部变量与外部函数示例

测试题库16

A.c

```
...
extern void fb();
extern void fc();
int x=0;
...
```

B.c

```
...
extern int x;
void fb()
{ ...}
...
```

C.c

```
...
extern int x;
void fc()
{ ...}
...
```

思考题：若B.c文件中删除外部变量声明语句“extern int x;”程序是否还能正确编译？

静态全局变量与静态函数

- 限制所定义的变量或函数只能在本文件中使用，而其他文件不能访问。
- 静态全局变量（Static Global Variable）
  - static <全局变量定义>;
- 静态函数（Static Function）
  - static <函数定义>

思考题修改例9.5　将A.c中变量x的定义改为static int x=0;
将B.c中函数fb()定义前加上static,
结果如何？

静态全局变量与静态函数

当一个团队共同完成一个项目时，不可避免地会出现大家使用了同样的标识符，这就可能造成重名问题。

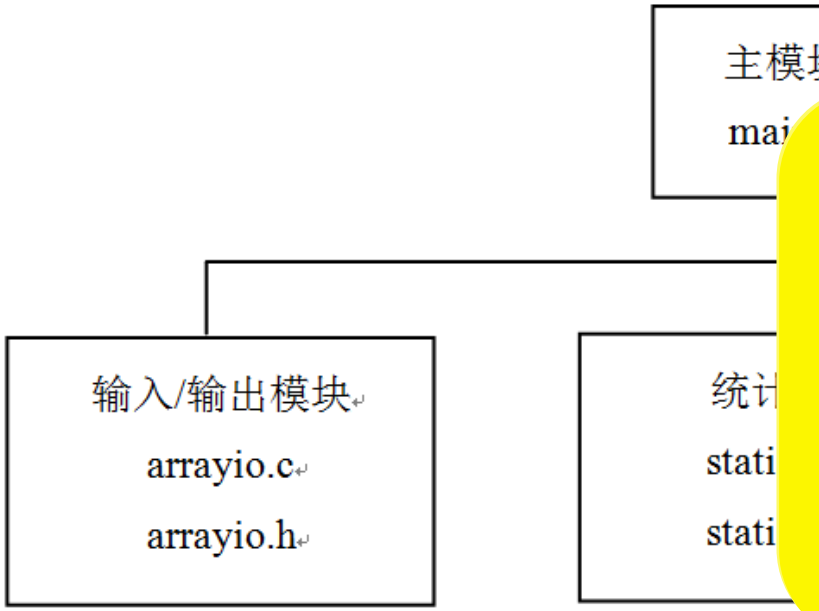
为避免重名，对于那些确定不需要共享的全局变量和禁止被其他文件调用的函数加上static，即定义为静态全局变量或静态函数，起到保护作用。

小结

- 外部变量与外部函数实现多文件之间的数据共享与协作
- 通过静态全局变量和静态函数实现数据和代码的保护

9.3 应用举例——多文件结构处理数组问题

- 例9.6设计一个多文件工程程序，实现对一维数组的输入、输出、统计、查找等。



模块化程序设计思想（Modular Programming）：

自顶向下、
逐步分解、
分而治之

本章小结

- 三种编译预处理指令
- 两种文件包含的区别
- 无参宏和带参宏的定义与替换
- 条件编译的语法与使用
- 外部变量与外部函数的定义、声明与使用
- 通过综合实例初步掌握模块的划分及多文件结构的实现

输入理想的程序

输出快乐的人生