

教育科学“十五”国家规划课题研究成果

C 程序设计

王柏盛 主 编
李万庆 贺洪江 副主编

高等教育出版社

内 容 提 要

本书全面介绍了 Turbo C 语言的基本概念,常量、变量、运算符和表达式,程序控制语句,函数,指针,结构、联合、枚举和定义类型,编译预处理命令,文件,字符屏幕和图形函数以及实用编程技术等内容。全书共分为 10 章。每章附有习题和实验,并精选了一部分全国计算机等级考试(二级 C 语言程序设计)的练习题,通过大量实例介绍 C 程序设计的思想、方法和技巧。

作者根据多年教学和科研积累的丰富经验,吸取当前一些 C 语言教材中的优点,增加了实用编程技术方面的内容,力求使本书体系合理、结构严谨、概念清晰、例题丰富、通俗易懂。

本书可作为高等院校程序设计课程的教材,也可供自学者使用或作为教师教学参考书。

与本书配套的辅助教材《C 程序设计习题题解》也同时出版。

前 言

C 语言是在国内外得到广泛使用的结构化程序设计语言,许多大型开发工具都以 C 语言作为基础语言。C 语言功能丰富、表达力强,使用方便灵活,目标程序效率高,可移植性好,既有结构化高级程序设计语言的优点,又有低级语言的绝大部分功能,它可以取代汇编语言,用来编写大型系统软件、工具软件和控制软件。

现在,C 语言已不仅为计算机专业工作者所使用,而且为广大工程技术人员所喜爱,许多人已经用它编写出很多优秀的应用软件。高等院校在非计算机专业普遍开设了 C 语言程序设计课程,大部分学生都选择 C 语言参加全国计算机等级考试。

本书全面介绍了 Turbo C 语言的基本概念,常量、变量、运算符和表达式,程序控制语句,函数,指针,结构、联合、枚举和定义类型,编译预处理命令,文件,字符屏幕和图形函数以及实用编程技术等内容。全书共 10 章,每章均附有习题和实验,并精选了一部分全国计算机等级考试(二级 C 语言程序设计)的练习题,通过大量实例介绍了 C 程序设计的思想、方法和技巧。

作者根据多年教学和科研积累的丰富经验,吸取当前一些 C 语言教材中的优点,大篇幅增加了字符屏幕、图形函数和实用编程技术方面的内容,力求使本书集教材、参考资料为一体,具有较强的实用性。本书的主要特点可以概括为以下几个方面:

(1) 体系合理、结构严谨、概念清晰、例题丰富,在章节安排上,更符合结构化程序设计语言的特点。

(2) 在程序设计技巧上有所创新,充分运用了 C 语言各种运算符和表达式的功能。各章设计了很多例程序并提供多种解法,以加强读者阅读能力、编程能力、调试能力和创新能力的培养。

(3) 增加了字符屏幕、图形函数及实用编程技术等章节,使本书在深度和广度上都得到了加强。

(4) 指出了 Turbo C 在表达式中存在的严重不一致性。

讲解本书约需 70 学时,其中上机实验 20 学时。不讲第九章和第十章需要 50 学时,其中包括 14 学时上机实验。有些章节,如第三章,有大量的例题,不少例题属于阅读、理解、加深题,学时不充分时可以跳过不讲。

由于程序设计语言是个有机的整体,各部分内容之间相互联系、渗透,因此,读者在使用本书时,应该在学习后面章节的同时经常复习前面的内容,以便加深理解。

与本书配套的辅助教材《C 程序设计题解》也同时出版。

本书第二、三、四、九、十章由王柏盛编写,第五、八章由李万庆编写,第一、六、七章和全部附录由贺洪江编写。全书由王柏盛统稿、整理。

本书在编写、出版过程中得到了河北工程大学有关领导的关心和支持,在此深致谢意。

由于作者的水平有限、经验不足,编写时间仓促,书中难免存在许多缺点不足之处,恳请广大读者和同行批评指正。

王柏盛
2003 年 12 月

目 录

| | |
|-----------------------------------|------|
| 第一章 C 语言概述 | (1) |
| 1.1 C 语言的起源 | (1) |
| 1.2 C 语言的特点 | (1) |
| 1.3 C 语言的词法 | (2) |
| 1.3.1 字符集 | (2) |
| 1.3.2 关键字 | (3) |
| 1.3.3 标识符 | (3) |
| 1.4 C 程序的组成和结构特点 | (4) |
| 1.4.1 程序举例 | (4) |
| 1.4.2 结构特点 | (6) |
| 1.5 C 程序的编辑、编译、连接和运行 | (7) |
| 1.5.1 C 源程序的编辑 | (7) |
| 1.5.2 C 源程序的编译和连接 | (7) |
| 1.5.3 Turbo C 的内存映射 | (7) |
| 1.5.4 C 源程序的调试过程 | (8) |
| 1.6 标准输入/输出函数 | (8) |
| 1.6.1 格式化输入/输出函数 | (8) |
| 1.6.2 非格式化输入/输出函数 | (14) |
| 习题一 | (17) |
| 实验一 Turbo C 源程序的编辑、编译、调试和运行 | (20) |
| 第二章 常量、变量、运算符和表达式 | (21) |
| 2.1 数据类型 | (21) |
| 2.2 常量 | (22) |
| 2.2.1 常量的数据类型 | (22) |
| 2.2.2 常量的表示方法 | (22) |
| 2.3 变量 | (23) |
| 2.3.1 变量的类型 | (23) |
| 2.3.2 变量的定义 | (23) |
| 2.3.3 变量的作用域 | (25) |
| 2.3.4 变量的存储类型 | (28) |
| 2.3.5 变量的初始化 | (32) |
| 2.4 数组 | (33) |

| | |
|-------------------------------------------|------|
| 2.4.1 数组的定义..... | (33) |
| 2.4.2 数组的引用..... | (34) |
| 2.4.3 数组的初始化..... | (35) |
| 2.4.4 应用举例..... | (37) |
| 2.5 指针 | (39) |
| 2.6 运算符和表达式 | (39) |
| 2.6.1 算术运算符和加 1、减 1 运算符 | (39) |
| 2.6.2 关系运算符、逻辑运算符及其表达式 | (42) |
| 2.6.3 按位运算符和位运算表达式..... | (43) |
| 2.6.4 特殊运算符及其表达式..... | (48) |
| 2.6.5 运算符优先顺序和结合性..... | (51) |
| 2.7 表达式的计算过程和数据类型转换 | (52) |
| 2.7.1 表达式的计算过程..... | (52) |
| 2.7.2 表达式中的类型转换..... | (55) |
| 2.7.3 程序举例..... | (57) |
| 2.8 综合举例 | (60) |
| 习题二 | (66) |
| 实验二 基本输入/输出函数和运算符、表达式 | (72) |
| 第三章 程序控制语句 | (73) |
| 3.1 C 语句概述..... | (73) |
| 3.1.1 C 程序结构 | (73) |
| 3.1.2 语句分类..... | (73) |
| 3.2 结构化程序基本结构 | (75) |
| 3.2.1 顺序结构..... | (75) |
| 3.2.2 选择结构..... | (75) |
| 3.2.3 循环结构..... | (76) |
| 3.3 顺序结构程序设计语句 | (77) |
| 3.4 分支结构程序设计语句 | (78) |
| 3.4.1 if 语句 | (78) |
| 3.4.2 switch 语句 | (85) |
| 3.5 循环结构程序设计语句 | (92) |
| 3.5.1 goto 语句以及用 goto 语句和 if 语句构成循环 | (92) |
| 3.5.2 while 语句 | (94) |
| 3.5.3 do while 语句 | (94) |
| 3.5.4 for 语句 | (95) |
| 3.5.5 循环的嵌套..... | (98) |
| 3.5.6 几种循环的比较..... | (98) |

| | |
|------------------------------------|-------|
| 3.5.7 程序举例 | (98) |
| 3.6 break 和 continue 语句 | (104) |
| 3.6.1 break 语句 | (104) |
| 3.6.2 continue 语句 | (105) |
| 3.6.3 程序举例 | (105) |
| 3.7 return 语句和 exit() 函数调用语句 | (107) |
| 3.7.1 return 语句 | (107) |
| 3.7.2 exit() 函数调用语句 | (107) |
| 3.8 综合举例 | (108) |
| 习题三 | (127) |
| 实验三(1) 分支结构程序设计 | (138) |
| 实验三(2) 循环结构程序设计 | (138) |
| 第四章 函数 | (139) |
| 4.1 函数的定义 | (139) |
| 4.1.1 定义形式 | (139) |
| 4.1.2 使用说明 | (140) |
| 4.1.3 应用举例 | (142) |
| 4.1.4 Turbo C 函数的扩展定义 | (142) |
| 4.2 函数的调用 | (144) |
| 4.2.1 调用形式 | (144) |
| 4.2.2 调用过程 | (145) |
| 4.2.3 调用条件 | (146) |
| 4.2.4 嵌套调用 | (147) |
| 4.3 函数间的数据传递 | (150) |
| 4.3.1 传值方式传递数据 | (151) |
| 4.3.2 传址方式传递数据 | (152) |
| 4.3.3 利用全局变量传递数据 | (153) |
| 4.3.4 处理结果在函数间的传递 | (154) |
| 4.4 函数与数组 | (154) |
| 4.5 递归函数 | (156) |
| 4.6 综合举例 | (160) |
| 习题四 | (173) |
| 实验四 函数 | (177) |
| 第五章 指针 | (178) |
| 5.1 指针变量的定义和初始化 | (178) |
| 5.1.1 指针的概念 | (178) |
| 5.1.2 指针变量的定义 | (180) |

| | |
|-------------------------------|-------|
| 5.1.3 指针变量的初始化 | (181) |
| 5.1.4 近程指针和远程指针 | (182) |
| 5.2 指针运算 | (182) |
| 5.2.1 取地址运算 | (182) |
| 5.2.2 赋值运算 | (183) |
| 5.2.3 取内容运算 | (183) |
| 5.2.4 指针的算术运算 | (184) |
| 5.2.5 关系运算 | (186) |
| 5.3 指针与数组 | (187) |
| 5.3.1 指向数组元素的指针变量的定义和引用 | (187) |
| 5.3.2 指向多维数组的指针变量 | (190) |
| 5.3.3 字符串的指针变量 | (192) |
| 5.4 指针和函数 | (196) |
| 5.4.1 用指针作为函数的参数 | (196) |
| 5.4.2 指向函数的指针变量 | (201) |
| 5.4.3 指针型函数 | (206) |
| 5.5 指针数组和多级指针 | (209) |
| 5.5.1 指针数组 | (209) |
| 5.5.2 指针的指针 | (211) |
| 5.5.3 指针数组作主函数的形参 | (212) |
| 5.6 程序举例 | (214) |
| 习题五 | (219) |
| 实验五 指针 | (227) |
| 第六章 结构、联合、枚举和定义类型 | (228) |
| 6.1 结构 | (228) |
| 6.1.1 结构的说明 | (228) |
| 6.1.2 结构变量的定义 | (229) |
| 6.1.3 结构成员的引用 | (232) |
| 6.1.4 结构变量的初始化 | (233) |
| 6.1.5 指向结构的指针 | (236) |
| 6.1.6 用指向结构的指针作为函数参数 | (238) |
| 6.1.7 结构型函数和结构指针型函数 | (242) |
| 6.1.8 动态数据结构 | (246) |
| 6.1.9 位域结构 | (249) |
| 6.2 联合 | (252) |
| 6.2.1 联合说明和联合变量的定义 | (252) |
| 6.2.2 联合变量的引用方式 | (253) |

| | |
|------------------------------------------------------|-------|
| 6.2.3 联合类型数据的特点 | (254) |
| 6.2.4 应用举例 | (256) |
| 6.3 枚举 | (258) |
| 6.4 定义类型 | (261) |
| 习题六 | (263) |
| 实验六 结构、联合、枚举 | (266) |
| 第七章 编译预处理命令 | (267) |
| 7.1 宏定义 | (267) |
| 7.1.1 不带参数的宏定义 | (267) |
| 7.1.2 带参数的宏定义 | (270) |
| 7.2 文件包含 | (275) |
| 7.3 条件编译 | (276) |
| 习题七 | (279) |
| 实验七 编译预处理命令 | (281) |
| 第八章 文件 | (282) |
| 8.1 文件概述 | (282) |
| 8.1.1 流和文件 | (282) |
| 8.1.2 标准设备文件的换向和管道连接 | (284) |
| 8.1.3 控制台输入/输出函数 | (287) |
| 8.2 文件类型指针 | (288) |
| 8.3 文件的打开与关闭 | (289) |
| 8.3.1 文件的打开(fopen()函数) | (289) |
| 8.3.2 文件的关闭(fclose()函数) | (291) |
| 8.4 文件结束检测及出错检测 | (291) |
| 8.4.1 feof()函数 | (291) |
| 8.4.2 ferror()函数 | (292) |
| 8.5 文件的读/写 | (292) |
| 8.5.1 fputc()函数和 fgetc()函数(putc()函数和 getc()函数) | (292) |
| 8.5.2 fread()函数和 fwrite()函数 | (295) |
| 8.5.3 fprintf()函数和 fscanf()函数 | (297) |
| 8.5.4 其他读/写函数 | (298) |
| 8.6 文件的定位 | (300) |
| 8.6.1 rewind()函数 | (300) |
| 8.6.2 fseek()函数 | (300) |
| 8.6.3 ftell()函数 | (301) |
| 8.7 非缓冲文件系统 | (302) |
| 8.7.1 open()、creat()和 close()函数 | (302) |

| | |
|------------------------------------------|-------|
| 8.7.2 read()和 write()函数 | (304) |
| 8.7.3 lseek()函数和 tell()函数 | (305) |
| 8.8 小结 | (305) |
| 习题八 | (306) |
| 实验八 文件 | (308) |
| 第九章 字符屏幕和图形函数 | (309) |
| 9.1 PC 图形适配器及其工作模式 | (309) |
| 9.2 字符屏幕函数 | (310) |
| 9.2.1 窗口 | (310) |
| 9.2.2 基本输入/输出函数 | (310) |
| 9.2.3 屏幕操作函数 | (311) |
| 9.2.4 字符属性控制函数 | (315) |
| 9.2.5 字符屏显状态函数 | (318) |
| 9.2.6 directvideo 变量 | (320) |
| 9.2.7 演示程序 | (320) |
| 9.3 Turbo C 的图形函数 | (321) |
| 9.3.1 图形模式的初始化 | (322) |
| 9.3.2 屏幕颜色的设置和清屏函数 | (325) |
| 9.3.3 基本图形函数 | (327) |
| 9.3.4 封闭图形的填充 | (332) |
| 9.3.5 有关图形视口和图形操作函数 | (337) |
| 9.3.6 图形模式下的文本输出 | (341) |
| 9.3.7 独立图形运行程序的建立 | (345) |
| 习题九 | (346) |
| 实验九 字符屏幕和图形函数 | (347) |
| 第十章 实用编程技术 | (348) |
| 10.1 Turbo C 库函数介绍 | (348) |
| 10.1.1 库文件的概念 | (348) |
| 10.1.2 Turbo C 提供的 BIOS、DOS 系统调用函数 | (350) |
| 10.1.3 日期和时间函数 | (363) |
| 10.1.4 字符串函数、数字字符串与数值的转换函数 | (367) |
| 10.1.5 动态内存分配函数、过程控制和数学运算函数 | (370) |
| 10.2 Turbo C 的存储模式 | (374) |
| 10.2.1 Turbo C 的存储模式 | (374) |
| 10.2.2 编译程序的内存模式选择 | (376) |
| 10.2.3 混合模式编程 | (376) |
| 10.2.4 Turbo C 的段修饰符 | (378) |

| | |
|-------------------------------------|-------|
| 10.3 Turbo C 集成开发环境下程序的调试 | (378) |
| 10.3.1 编译时的常见错误 | (378) |
| 10.3.2 连接时的常见错误 | (379) |
| 10.3.3 运行时的常见错误 | (379) |
| 10.4 Turbo C 的命令行编译 | (380) |
| 10.5 Turbo C 中汉字的使用 | (382) |
| 10.5.1 汉字操作系统下汉字输入/输出的程序编制 | (382) |
| 10.5.2 非汉字操作系统下汉字的使用 | (385) |
| 10.6 Turbo C 和汇编程序的接口 | (396) |
| 10.6.1 Turbo C 调用汇编子程序 | (396) |
| 10.6.2 Turbo C 行间嵌入汇编 | (398) |
| 10.7 Turbo C 2.0 集成开发环境的安装和使用 | (401) |
| 10.7.1 Turbo C 2.0 软盘内容简介 | (401) |
| 10.7.2 Turbo C 2.0 的安装和启动 | (401) |
| 10.7.3 Turbo C 2.0 集成开发环境的使用 | (402) |
| 10.7.4 Turbo C 的配置文件 | (410) |
| 附录 | (412) |
| 附录一 常用字符与 ASCII 码对照表 | (412) |
| 附录二 C 语言中的关键字 | (413) |
| 附录三 运算符和优先级 | (413) |
| 附录四 C 语言常用语法提要 | (414) |
| 附录五 Turbo C 常用库函数表 | (419) |
| 附录六 键盘扩展码表 | (438) |
| 参考文献 | (439) |

第七章 编译预处理命令

C 语言和其他高级语言的一个重要区别是它具有编译预处理功能。C 语言的编译预处理功能是通过预处理程序实现的。C 编译预处理程序负责分析和处理以“#”开头的编译预处理命令。C 语言的预处理命令主要有宏替换、文件包含和条件编译。由于它们是在编译系统的第一遍扫描,即词法和语法之前进行的,所以称为预处理命令。

一个 C 源文件一般要经过编辑、预处理、编译、汇编和连接五个阶段。

从语法上讲,预处理命令和 C 语言的其他成分无关,它们可以出现在程序中的任何地方,一般宏替换和文件包含应出现在文件的开头。预处理命令的作用范围仅限于说明它们的那个文件,出了那个文件它们就失去了作用。

准确地使用 C 语言的预处理功能可以编写出易读、易改、易于移植和调试的 C 程序,有利于工程的模块化设计。

7.1 宏 定 义

以 `#define` 作为标志的预处理命令称为宏定义命令。该命令用于定义符号常量和定义带参数的宏。

7.1.1 不带参数的宏定义

不带参数的宏定义,即用一个指定的标识符(即名字)来代表一个字符串,它的一般形式是:

```
#define 标识符 字符串
```

其中, `#define` 是预处理宏替换命令,标识符一般由大写字母组成,以便与程序中的变量名和函数名相区别;字符串为 C 语言字符集中的任意字符序列,字符串不带双引号。`#define`、标识符、字符串之间以空格分隔,末尾不加分号。每个预处理命令行占一行。例如:

```
#define PI 3.141592654
```

包含该命令的文件在预处理过程中,凡是以 `PI` 作为标记出现的地方都用 `3.141 592 654` 来替换。这种方法使用户能以一个简单的名字代替一个长的字符

串,尤其是当程序中多次用到 PI 时,其好处更为突出,而且这样做对于修改、阅读和移植程序都是十分方便的。

由于 #define 命令只进行简单的字符串替换,因此把 #define 称为“宏定义”命令,把标识符称为“宏名”,在预处理时将宏名替换成字符串的过程称为:“宏展开”。

又如,对于熟悉 Pascal 或 ALGOL 语言的读者来说,可以用:

```
#define BEGIN {  
#define END }
```

进行宏定义。这样使得 BEGIN 与“{”,END 与“}”具有相同的含义,可以将一个 C 语言程序描述成类似 Pascal 或 ALGOL 语言程序的形式:

```
...  
BEGIN  
...  
BEGIN  
...  
END  
...  
END
```

这样可以用阅读 Pascal 或 ALGOL 语言程序的习惯来阅读 C 语言程序。

再如,由于 C 语言中数据没有逻辑类型只是用零和非零来表示逻辑值,为了遵从人们的习惯,可以利用宏定义来解决这个问题:

```
#define TURE 1  
#define FALSE 0
```

在符号常量定义中,对于一些常量有一定的习惯约定。例如,当程序中使用 0 和 1 作为条件判别时,常把它们定义为:

```
#define YES 1  
#define NO 0
```

还有,将 EOF 定义为文件结束标志,把 NULL 定义为空字符等等都可以减少程序对机器的依赖性。利用宏定义可以增加程序的移植性,这是一个很重要的方法。

在 C 语言中,数组不允许动态定义,这影响到程序的通用性,如果使用宏定义,就可将程序稍加修改便能适应不同情况。例如:

```
#define M 100  
...  
int array[M];  
...
```

则只需要修改宏定义就能达到修改数组长度的目的。

【例 7.1】

```
#define PI 3.1415926
main()
{
    float l,s,r,v;
    printf( "Input radius : ");scanf( "%f" &r );
    l= 2* PI*r;
    s= PI*r*r;
    v=4.0/3* PI*r*r*r;
    printf( "L = %10.4f\nS = %10.4f\nV = %10.4f\n", l,s,v );
}
```

运行情况如下：

```
Input radius #
L = 25.1328
S = 50.2655
V = 150.7966
```

说明：

- (1) 宏名一般用大写字母表示, 以与变量名、函数名相区别。
- (2) 宏定义不是 C 语句, 末尾不加分号, 否则分号作为字符串的一部分一起进行替换。
- (3) 宏定义只是用宏名代替一个字符串, 不作语法检查。
- (4) #define 命令一般出现在文件的开头, 其有效范围从它出现起到本源文件的结束。可以用 #undef 命令终止宏定义的作用域。
- (5) 在进行宏定义时, 可以引用已定义过的宏名, 可以层层置换。

【例 7.2】

```
#define R 3.0
#define PI 3.1415926
#define L 2* PI* R
#define S PI* R* R
main()
{
    printf( "L = %f\nS = %f\n", L, S );
}
```

运行情况如下：

```
L = 18.849556
```

```
S = 28.274333
```

(6) 程序中用双引号括起来的字符串中若含有宏名则在宏展开时不进行宏置换。例如，上例中的 L，一个在双引号内没有进行宏置换，一个在双引号外进行了宏置换。

7.1.2 带参数的宏定义

带参数宏定义的一般形式是：

```
#define 宏名(参数表) 字符串
```

其中，参数表中的参数类似于函数中的形参，字符串中包含括号中所指定的参数。例如：

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

```
...
```

```
area = S(3,2);
```

定义矩形面积 S，a 和 b 是边长。在程序中用了 S(3,2)，把 3,2 分别代替宏定义中的形参 a,b，即用 3*2 代替 S(3,2)。因此该赋值语句展开为：

```
area = 3*2;
```

对带参的宏定义是这样展开置换的：在程序中如果有带实参的宏（例如 S(3,2)），则按 #define 命令行中指定的字符串从左到右进行置换。如果字符串中包含宏中的形参（如 a,b），则将程序语句中相应的实参（可以是常量、变量或表达式）代替形参，如果宏定义中的字符串中的字符不是参数字符（如 a*b 中的 * 号），则保留。这样就形成了置换的字符串。

【例 7.3】

```
#define PI 3.1415926
#define S(r) PI*r*r
main()
{
    float a,area;
    a=3.6,area=S(a);
    printf("R = %f\nAREA = %f\n",a,area);
}
```

运行结果如下：

```
R = 3.600000
```

```
AREA = 40.715038
```

赋值语句 `area = S(a)` 经宏展开后为

```
area = 3.1415926 * a * a ;
```

说明：

(1) 宏定义时,宏名与带参的圆括号之间不能有空格,否则将空格后的字符都作为替代字符串的一部分。这就成了不带参的宏定义了。

(2) 对带参的宏的展开只是将语句中的宏名后面括号内的实参字符串代替 `#define` 命令行中的形参,需要注意宏展开后的正确性。例 7.3 语句中有 `S(a)`,在展开时,找到 `#define` 命令行中的 `S(r)`,将 `S(a)` 中的实参 `a` 代替宏定义中的字符串“`PI * r * r`”中的形参 `r`,从而得到 `PI * a * a`。这是容易理解的,也不会发生什么问题。但是,如果有以下语句：

```
area = S(a + b) ;
```

这时把实参 `a + b` 代替 `PI * r * r` 中的形参 `r`,成为

```
area = PI * a + b * a + b ;
```

请注意在 `a + b` 外面没有括号,显然这与程序设计者的原意不符。原希望得到

```
area = PI * (a + b) * (a + b) ;
```

为了得到这个结果,应该在定义时在字符串中的形参外面加上括号。即

```
#define S(r) PI * (r) * (r)
```

在对 `S(a + b)` 进行宏展开时,将 `a + b` 代替 `r` 就成了

```
PI * (a + b) * (a + b)
```

这就达到了目的。

除了字符串中的形参要用圆括号括起来外,整个字符串最好也用圆括号括起来。例如

```
#define SQUARE(x) (x) * (x)          /* 定义的x平方 */
main()
{
    printf( "%f\n", 27.0/SQUARE(3.0) );
}
```

输出的结果确是 27.000000。因为该程序在预处理后成为：

```
main()
{
    printf( "%f\n", 27.0/(3.0) * (3.0) );
}
```

```
    }
```

由于/和*是同一优先级,从左到右进行运算,上面的表达式就等价于:

```
(27.0/3.0) * 3.0
```

所以会输出 27.000000。解决问题的办法是在宏定义中,把字符串整个用圆括号括起来。

```
#define SQUARE(x) ((x) * (x))
```

(3) 带参宏定义与函数相似但不同:

函数调用时先求实参表达式的值,然后代入形参。而使用带参的宏只是进行简单的字符替换。

函数调用是在程序运行时处理的,分配临时的内存单元。而宏展开则是在预编译时进行的,在展开时并不分配内存单元,不进行值的传递处理,也没有返回值的概念。

【例 7.4】 用函数和宏定义分别求 1 到 9 平方值的程序。

用函数来做:

```
main()
{
    int i = 1;
    while(i < 10) printf( "%d \t square(i++) );
}
square(int n)
{
    return(n*n);
}
```

运行结果如下:

```
1    4    9    16    25    36    49    64    81
```

用宏定义来做:

```
#define SQUARE(n) (n * n)
main()
{
    int i = 1;
    while(i < 10) printf( "%d \t SQUARE(i++) );
}
```

运行结果如下: