

最新Microsoft QuickC

2.50版

使用大全



(共四册)

周杉 王忠 编译
晓夏 武仁

■ 语言参考手册

中国科学院希望高级电脑技术公司

2

最新Microsoft QuickC

使 用 大 全

（V2.50版）

- 安装和运行·成套工具（之一）
- 语言参考手册（之二）
- 库函数集锦（之三）
- 库函数集锦（之四）

中国科学院希望高级电脑技术公司
一九九〇年十一月

目 录

引言	1
§ 0.1 本书内容	1
§ 0.2 使用示例程序	1
§ 0.3 本手册所使用的编程风格	2
第一部分 学习 C	3
第一章 C 程序解析	3
§ 1.1 一个典型的 C 程序	3
§ 1.2 注释	3
§ 1.3 语句	4
§ 1.4 关键字和名字	4
§ 1.5 预处理指令	4
§ 1.6 函数	4
§ 1.7 调用函数	5
§ 1.8 声明及初始化变量	5
§ 1.9 外部变量与局部变量	5
§ 1.10 函数原型	6
§ 1.11 关于 printf	6
第二章 函数	8
§ 2.1 函数和结构化程序设计	8
§ 2.2 main 函数	8
§ 2.3 函数的位置与可见性	9
§ 2.4 函数定义和原型	10
§ 2.5 调用一个函数	10
§ 2.6 传递变元	12
§ 2.7 变元与参数	13
§ 2.8 给参数赋值	13
§ 2.9 值传递	14
§ 2.10 从函数中返回值	16
§ 2.11 使用返回值	17
§ 2.12 声明函数的返回类型	18
§ 2.13 函数原型	18
§ 2.13.1 原型化无参数的函数	19
§ 2.13.2 原型化有不定参数的函数	19

§ 2.14 函数声明与定义的旧式风格	19
第三章 流程控制	22
§ 3.1 循环: while、do 和 for	22
§ 3.1.1 while 语句	22
§ 3.1.2 do 语句	23
§ 3.1.3 for 循环	25
§ 3.2 判断语句: if、else、switch、break、continue 和 goto	27
§ 3.2.1 if 语句	27
§ 3.2.2 else 子句	29
§ 3.2.3 switch 语句	30
§ 3.2.4 break 语句	32
§ 3.2.5 continue 语句	34
§ 3.2.6 goto 语句	35
第四章 数据类型	36
§ 4.1 基本数据类型	36
§ 4.1.1 说明基本类型	36
§ 4.1.2 说明变量	38
§ 4.1.3 说明常量	38
§ 4.2 聚集数据类型	40
§ 4.2.1 数组	40
§ 4.2.2 联合	53
第五章 高级数据类型	55
§ 5.1 可见性	55
§ 5.1.1 局部变量	55
§ 5.1.2 外部变量	57
§ 5.1.3 多个源文件中的可见性	58
§ 5.1.4 函数的可见性	59
§ 5.2 生命期	59
§ 5.2.1 延长局部变量的生命期	60
§ 5.3 转换数据类型	61
§ 5.3.1 数据类型的级别	61
§ 5.3.2 升级与降级	61
§ 5.3.3 自动类型转换	62
§ 5.3.4 强制类型转换	64
§ 5.4 寄存器变量	65
§ 5.5 用 typedef 对现存类型重命名	66
§ 5.6 枚举类型	66

第六章 操作符	68
§ 6.1 C 操作符介绍	68
§ 6.1.1 算术操作符	68
§ 6.1.2 关系操作符	69
§ 6.1.3 赋值操作符	69
§ 6.2 C 语言独特的操作符	70
§ 6.2.1 加 1 和减 1 符	70
§ 6.2.2 位操作符	71
§ 6.2.3 逻辑操作符	73
§ 6.2.4 地址操作符	74
§ 6.2.5 条件操作符	74
§ 6.2.6 sizeof 操作符	75
§ 6.2.7 逗号操作符	75
§ 6.2.8 基数操作符	75
§ 6.3 操作符优先级	75
第七章 预处理指令	78
§ 7.1 #include 指令	78
§ 7.1.1 说明包含文件	79
§ 7.2 #define 和 #undef 指令	79
§ 7.2.1 简单文本替换	80
§ 7.2.2 似函数的宏	80
§ 7.2.3 #undef 指令	81
§ 7.3 条件指令	81
§ 7.3.1 defined 操作符	82
§ 7.4 编译指令	83
第八章 指针	84
§ 8.1 使用指针	84
§ 8.2 指向简单变量的指针	84
§ 8.2.1 声明指针变量	85
§ 8.2.2 指针存储方式	85
§ 8.2.3 初始化指针变量	86
§ 8.2.4 使用指针变量	87
§ 8.2.5 指针基础知识小结	87
§ 8.3 数组指针	88
§ 8.3.1 数组及指针运算	89
§ 8.3.2 指针比较	90
§ 8.3.3 重看 PARRAY.C	90
§ 8.4 指针和串	91

§ 8.5 传递指针至函数	93
§ 8.5.1 传送地址常量与传送指针变量	95
§ 8.6 指针数组	96
§ 8.7 本章回顾	99
 第九章 高级指针	 100
§ 9.1 指向指针的指针	100
§ 9.2 数组与指针的等价性	101
§ 9.3 获取命令行参数	104
§ 9.4 空指针	105
§ 9.5 指向结构的指针	105
§ 9.6 指向函数的指针	108
§ 9.6.1 把函数指针作为参数传递	109
§ 9.7 有关指针的几点说明	110
 第十章 程序设计中的错误	 111
§ 10.1 操作符问题	111
§ 10.1.1 混淆赋值和相等操作符	111
§ 10.1.2 混淆操作符优先级	112
§ 10.1.3 混淆结构元素操作符	112
§ 10.2 数组问题	113
§ 10.2.1 数组下标错	113
§ 10.2.2 在处理数组时忽略了数组下标	113
§ 10.2.3 超出数组边界	114
§ 10.3 字符串问题	114
§ 10.3.1 混淆字符常数与字符串	114
§ 10.3.2 忘记了字符串结尾的空字符	115
§ 10.3.3 忘记为字符串分配内存	115
§ 10.4 指针问题	116
§ 10.4.1 用错误的地址操作符来初始化指针	116
§ 10.4.2 用错误的类型声明了一个指针	117
§ 10.4.3 使用悬挂指针	117
§ 10.5 库函数问题	118
§ 10.5.1 没有检查库函数的返回值	118
§ 10.5.2 与库函数重名	119
§ 10.5.3 忘记包含声明库函数的包含文件	119
§ 10.5.4 调用 scanf 时忽略了地址操作符	120
§ 10.6 宏问题	120
§ 10.6.1 忽略了宏参数的括号	121
§ 10.6.2 在宏参数中使用递增和递减操作符	121

§ 10.7 混合性问题	123
§ 10.7.1 不匹配的 if 和 else 语句	123
§ 10.7.2 放错了分号的位置	124
§ 10.7.3 在 DOS 路径中忽略了双反斜杠	125
§ 10.7.4 忽略了开关语句中的 break 语句	126
§ 10.7.5 混淆有符号和无符号值	126
 第二部分 使用 C	 128
 第十一章 输入和输出	 129
§ 11.1 输入输出流	129
§ 11.2 屏幕和键盘输入/输出	129
§ 11.2.1 操作和打印字符串	129
§ 11.2.2 打印数值	133
§ 11.2.3 使用 scanf 进行键盘输入	136
§ 11.3 标准磁盘输入/输出	139
§ 11.3.1 创建文件和向文本文件写	139
§ 11.3.2 打开文件以便二进制方式读	141
§ 11.3.3 二进制和文本文件	142
§ 11.3.4 数值型变量的文本格式	144
§ 11.3.5 使用二进制格式	148
§ 11.4 低级输入和输出	151
§ 11.4.1 低级读和低级写	152
 第十二章 动态存储分配	 155
§ 12.1 为什么要分配?	155
§ 12.2 存储分配基础	155
§ 12.2.1 分配存储块的准备工作	157
§ 12.2.2 指定分配块的大小	158
§ 12.2.3 图示说明	158
§ 12.2.4 对 malloc 返回地址赋值	159
§ 12.2.5 检测 malloc 函数的返回值	160
§ 12.2.6 访问一分配存储块	160
§ 12.2.7 为不同的数据类型分配存储块	161
§ 12.2.8 采用 free 函数释放存储块	162
§ 12.3 特定的存储分配函数	163
§ 12.3.1 calloc 函数	163
§ 12.3.2 realloc 函数	163
§ 12.4 避免麻烦	163

第十三章 图形	165
§ 13.1 图形方式	165
§ 13.1.1 检测当前视频模式	165
§ 13.1.2 设置视频方式	166
§ 13.1.3 编制图形程序	167
§ 13.1.4 使用彩色图形方式	173
§ 13.1.5 使用彩色视频文本方式	181
§ 13.2 文本坐标	182
§ 13.3 图形坐标	183
§ 13.3.1 物理屏幕	183
§ 13.3.2 视口坐标	186
§ 13.3.3 窗口中的实际坐标	186
第十四章 表示图形	193
§ 14.1 术语	193
§ 14.2 表示图形程序结构	195
§ 14.3 五个图形程序例子	196
§ 14.4 色板	206
§ 14.4.1 颜色库	206
§ 14.4.2 格式库	207
§ 14.4.3 模式库	207
§ 14.4.4 字符库	209
§ 14.5 通用表示图形	209
§ 14.5.1 图形环境	209
§ 14.5.2 <code>titletype</code>	210
§ 14.5.3 <code>axistype</code>	211
§ 14.5.4 窗口类型	213
§ 14.5.5 <code>legendtype</code>	214
§ 14.5.6 <code>chartenv</code>	215
§ 14.6 表示图形函数概述	216
第十五章 字体	218
§ 15.1 QuickC 字体	218
§ 15.2 使用 QuickC 字体库	220
§ 15.2.1 注册字体	220
§ 15.2.2 设置当前字体	220
§ 15.2.3 显示文本	222
§ 15.3 例题程序	222
§ 15.4 一些提示	224

第十六章 内部汇编	226
§ 16.1 内部汇编的优点	226
§ 16.2 <code>_asm</code> 关键字	226
§ 16.3 在 <code>_asm</code> 块中使用汇编语言	227
§ 16.3.1 指令集	227
§ 16.3.2 表达式	227
§ 16.3.3 <code>Data</code> 伪指令及操作符	227
§ 16.3.4 <code>EVEN</code> 和 <code>ALIGN</code> 伪指令	227
§ 16.3.5 宏	228
§ 16.3.6 段访问	228
§ 16.3.7 类型和变量大小	228
§ 16.3.8 注释	228
§ 16.3.9 用 CodeView 调试器来调试	229
§ 16.4 在 <code>_asm</code> 块中使用 C	229
§ 16.4.1 使用操作符	229
§ 16.4.2 使用 C 符号	229
§ 16.4.3 访问 C 数据	230
§ 16.4.4 编写函数	231
§ 16.5 使用和保存寄存器	232
§ 16.6 跳转到标号	232
§ 16.7 定义 <code>_asm</code> 块为 C 宏	233
§ 16.8 优化	235
附录 A C 语言指南	236
§ A.1 一般语法	236
§ A.1.1 用户定义名	236
§ A.1.2 关键字	236
§ A.1.3 函数	237
§ A.2 流程控制	237
§ A.2.1 <code>break</code> 语句	237
§ A.2.2 <code>continue</code> 语句	238
§ A.2.3 <code>do</code> 语句	238
§ A.2.4 <code>for</code> 语句	238
§ A.2.5 <code>goto</code> 语句	239
§ A.2.6 <code>if</code> 语句	239
§ A.2.7 <code>return</code> 语句	240
§ A.2.8 <code>switch</code> 语句	240
§ A.2.9 <code>while</code> 语句	241
§ A.3 数据类型	241
§ A.3.1 基本数据类型	241

§ A.3.3 高级数据类型	244
§ A.4 操作符	245
§ A.5 预处理指令	247
§ A.6 指针	249
附录 B C 库函数指南	250
§ B.1 C 运行库概况	250
§ B.2 缓冲区处理例程	252
§ B.3 字符分类与转换例程	253
§ B.4 数据转换例程	254
§ B.5 错误信息例程	255
§ B.6 图形 1: 低级图形例程	256
§ B.6.1 配置方式与环境	256
§ B.6.2 设置坐标	257
§ B.6.3 设置色板	259
§ B.6.4 设置属性	259
§ B.6.5 输出图像	260
§ B.6.6 输出文本	264
§ B.6.7 传递图像	265
§ B.7 图形 2: 表示图形例程	266
§ B.8 图形 3: 字体显示例程	269
§ B.9 输入、输出例程	270
§ B.9.1 流式例程	270
§ B.9.2 低级例程	276
§ B.9.3 控制台与端口 I/O 例程	278
§ B.10 数学例程	279
§ B.11 内存分配例程	283
§ B.12 进程控制例程	284
§ B.13 搜索与排序例程	285
§ B.14 字符串处理例程	286
§ B.15 时间例程	289

引 言

自从 Microsoft 在 1987 年推出 QuickC C Compiler 1.0 版以来, QuickC 用户要求更多地了解 C 程序设计语言。《语言参考手册》满足了这些要求,特别是满足了那些已有一定编程经验但还未学过 C 人们的要求。

§ 0.1 本书内容

《语言参考手册》假定你已具有一些编程经验,但还不熟悉 C。因此,本书不解释诸如为什么程序循环是有用的这种基本编程思想,而是认为你已一般地了解了循环,只是现在想来学习怎样用 C 语言实现它们。本书内容如下:

- 第一部分“学习 C”:包括了诸如函数及数据类型这样的基本问题。本部分中的诸章应按顺序从头到尾阅读。

- 第二部分“使用 C”:包括了诸如输入/输出和图形学这样的实际编程问题。本部分是按各个问题而组织的,所以你不必按照任何特定顺序来进行阅读。

- 附录 A “C 语言指南”:总结了 C 语言的 QuickC 实现。你可在读完第一部分并对 C 有些了解之后将此附录作为一快速参考。

- 附录 B “C 库指南”:总结了 QuickC 运行库中最常使用的函数。该附录主要是当你未在使用 QuickC 时作为浏览而用的。当在 Quickb 环境中,可使用 Microsoft QuickC Advisor(联机帮助)来得到有关 C 语言性质及运行库函数的信息。

注意,以后用术语“OS/2”来表示 OS/2 系统——Microsoft 操作系统/2(Ms OS/2)及 IBM OS/2。类似地,术语“DOS”既表示 MS-DOS,也表示 IBM DOS。只是在说明为某系统所独特的性质时才给出具体系统名。

§ 0.2 使用示例程序

本书中的示例程序可通过联机帮助而得到。这就使你可在阅读时装入、运行及尝试示例程序。

为装入一例子,必须使用 QuickC 环境:在 Help 菜单中选择 Contents,然后选择本书的标题。找到所希望的程序,然后用 QuickC 的 Copy 和 Paste 函数将它拷贝至编辑器中。

在将一程序拷贝至 QuickC Editor 中后,就可以将它看成是任何一个 C 的源程序,从而可对之编译或编辑,及将它存至磁盘等等。

QuickC 联机帮助包括了本书所有有意义的例子(它不包括代码段及一些很短的程序)。在联机帮助中的每个程序都以如下形式的一行语句开始:

```
/* POINTER.C: Demonstrate pointer basics. */
```

该行包括了程序名及程序功能的简单介绍。含此行的程序在联机帮助中被命名为 POINTER.C。

联机帮助中的所有例子编译时都不会在第三警告层上出错,QuickC 在此层上进行最严格的出错检查。在此层上,有些例子会产生下面这样无害的警告:

C4103: 'main' : function definition used as prototype

C4035: 'main' : no return value

C4051: data conversion

你可在一较低的警告层上进行编译以取消这些警告。

§ 0.3 本手册所使用的编程风格

C 语言源代码的格式具有很人的灵活性。本书中所用的风格是为了使程序可读性好，但在编写自己的程序时却不必用此风格。下面是本书中对示例程序所用风格的概要：

- 每个语句或函数都单独成行。
- 变量名及函数名都用小写。诸如 TRUE 或 FALSE 这样的符号常量名用大写。
- 如果一函数无参数，那么函数名后的一中间无空格的开括号及闭括号前各有一空格：`printf( "Number = %i", num_penguins );`
- 诸如加和减这样的二元操作符前后都是一空格：`3 + 5`
- 如果用括号来控制操作符的优先级，在开括号后及闭括号前就不用空格：`(3 + 5) * 2。`
- 开及闭花括号都位于控制关键字的第一个字符下方，其中的块缩进 3 个空格：

```
if( a == b )
{
    c = 50;
    printf( "%i\n", a );
}
```

第一部分 学习 C

第一章 C 程序解析

作为一个有经验的程序员，你可能会想立即投身于 C，但在此之前，你应该了解所有 C 程序的基本模型。本章剖析了一个 C 程序，但没有陷于规则的形式定义或例外情形中。

这里的讨论围绕一个不大的名为 VOLUME.C 的典型 C 程序。为习惯 C 程序的形式以及组成它们的基本成分，在阅读时应常常参考 VOLUME.C。

§ 1.1 一个典型的 C 程序

VOLUME.C 是一个计算一球体体积并打印下列结果于屏幕上的简单程序：

Volume: 113.040001

象本书中所有的典型程序一样，VOLUME.C 也在 QuickC 的联机帮助中。“引言”介绍了怎样装入典型程序。图 1.1 说明了 VOLUME.C 程序。

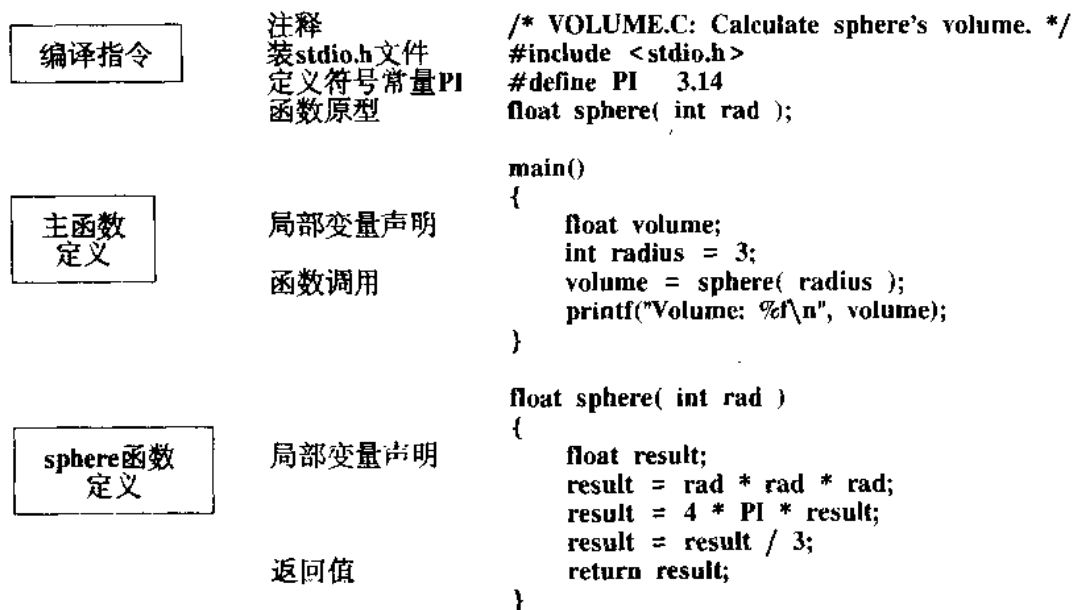


图 1.1 VOLUME.C 程序

§ 1.2 注释

VOLUME.C 的第一行是一注释：

```
/* VOLUME.C: Calculate sphere's volume. */
```

在 C 中，注释以 `/*` 开始，以 `*/` 结束。由于 C 只有很少的关键字，注释在使程序可读性

好这一方面就起了重要作用。

注释不可以嵌套(将一个注释放在另一个注释里面)。下面这行含有语法错误:

```
/* Error! /* You can't */ nest comments in C. */
```

§ 1.3 语句

C 语句总是以分号结束。例如 VOLUME.C 中的语句:

```
result = 4 * PI * result;
```

还可以用花括号将一组语句括起来以形成一个“语句块”。语句块含诸如一函数体内的各语句这样的相关语句。

C 语言忽略源程序中的空格, 所以你几乎可以以任意风格来安排程序。但是通常, 典型的 C 程序中是每行一个语句, 花括号垂直对应, 其中的语句是缩进的。引言中描述了本书所使用的编程风格。

§ 1.4 关键字和名字

C 是一种对大小写敏感的语言(即区别大写字母与小写字母)。所有 C 的关键字都用小写给出; 联机帮助中含有 C 关键字的一个完整集合。

你可以以大小写的任意组合来声明名字, 但是许多程序员更喜欢用小写来表示变量和函数名, 用大写来声明符号常量。(一个符号常量是一表示常数的描述名。在 VOLUME.C 中, PI 就是一符号常量)。

§ 1.5 预处理指令

一个程序中并非每行语句都是可执行的。程序可以包含“预处理指令”——用于 QuickC 编译器的命令。这样的指令以 # 开始, 结尾处不加分号。

VOLUME.C 的第二、三行就是预处理指令。#include 指令告诉 QuickC 在编译 VOLUME.C 时插进文件 stdio.h:

```
#include <stdio.h>
```

STDIO.H 是 QuickC 提供的许多包含文件中的一个。包含文件含有为 C 库函数所需的声明及定义(“库函数”是和 QuickC 一起提供而非由你编写的)。在程序 VOLUME.C 中, printf 库函数需要来自 stdio.h 包含文件中的信息。

#define 指令定义一名为 PI 的符号常量:

```
#define PI 3.14
```

这样, 每当 PI 在程序后面出现时, QuickC 就用 3.14 代替。符号常量可以是字母、数字或其它字符的任意组合。

§ 1.6 函数

函数是 C 的基本组块。每个 C 程序都至少有一个函数, 每个可执行的 C 语句都出现在某函数内部。简单地说, 一函数就是完成某特定任务并常常将一值返回至调用它的语句上的一组语句。

C 函数的目的与 Quick Pascal 的过程及函数, 或者 BASIC 的 SUB 及 FUNCTION 过程相同。利用它们可以写出组织良好的程序。

C 还用函数来完成所有的输入与输出(I/O)。与其它高级语言不同, C 无诸如 PRINT 和 READ 这样的语句, 所有的 I/O 都是通过调用 printf 这样的库函数而完成的。

VOLUME.C 程序含两个函数: main 和 sphere。每个 C 程序的主要执行部分本身就是一个名为 main 的函数, 该函数标志着执行的开始与结束。当运行 VOLUME.C 时, 执行从 main 函数的头部开始, 在 main 的尾部结束。

§ 1.7 调用函数

函数可在一程序的任何地方被调用, 它们既能接受值, 也能返回值。传递给函数的一个值被称为一个变元。

程序 VOLUME.C 含两个函数调用, 一个是调用库函数 printf, 另一个是调用程序中定义的函数 sphere。下列语句调用 printf 函数。

```
printf( "Volume: %f\n", volume );
```

该语句将两个变元传递给 printf: "Volume: %f\n"提供了文本及格式化信息; volume 给出了一数值。

C 中一函数不必一定要返回一值, 它可以象 QuickPascal 的函数那样返回一值, 也可以象 QuickPascal 的过程那样什么也不返回。

当函数有返回值时, 该值通常被赋给变量。VOLUME.C 中的下列语句调用函数 sphere 并将其返回值赋给变量 volume:

```
volume = sphere( radius );
```

函数用关键字 return 来返回一值。在 VOLUME.C 中, 函数 sphere 的最后语句将变量 result 的值返回至调用 sphere 的语句中:

```
return result
```

§ 1.8 声明及初始化变量

对 C 程序中的每个变量都必须说明它的名字和类型从而加以“声明”。如果用到了一个未声明变量, QuickC 在编译程序时显示出一错误信息。

VOLUME.C 中的下列语句声明了一个 float(浮点数)类型的名为 volume 的变量。

```
float volume;
```

在声明了一变量后就应在使用它前将其初始化, 即赋给它一值。未初始化的变量可有任意值, 所以使用它们是危险的。VOLUME.C 程序通过赋给变量 volume 一个来自函数调用的返回值而初始化了 volume:

```
volume = sphere( radius );
```

还可以在声明变量的同时对其初始化。VOLUME.C 的下列语句声明了变量 radius 为一 int(整数)类型变量, 并将其初始化为 3:

```
int radius = 3;
```

§ 1.9 外部变量与局部变量

声明变量发生在什么位置就决定了该变量在何处是可见的。在所有函数之外声明的变量是“外部的”。你可在程序中任何地方引用它(外部变量在一些其它语言中也叫全局变量)。

在一函数花括号以内声明的变量就是“局部的”。你只可在该函数内引用它。在 VOLUME.C 中, 变量 **result** 是在函数 **sphere** 内声明的:

```
float sphere( int rad )
{
    float result;
    .
    .
    .
}
```

由于它是局部于函数 **sphere** 的, 所以变量 **result** 就不能在 VOLUME.C 中的其它地方加以使用。将变量置成局部的可以最大地降低变量值可能会被程序的其它部分不小心地改变的危險。

当函数接受变元时, 这些变元就成为该函数内的局部变量。函数 **sphere** 需要一个变元, 命名它为 **rad**。在该函数内, **rad** 就是一局部变量。

§ 1.10 函数原型

函数声明与变量声明相差无几。函数声明——通常被称为“原型”——可使 QuickC 进行类型检查。给出原型中的信息, QuickC 就能在以后每次使用函数时加以检查以确信你传递了正确的变元数目与类型以及使用了正确的返回类型。

一个函数原型给出了如下信息:

- 函数名
- 函数返回值的类型
- 函数所需的变元表

VOLUME.C 程序包含了函数 **sphere** 的原型:

```
float sphere( int rad );
```

该原型表明函数 **sphere** 返回一浮点(float)值, 并且需要一整数(int)型变元。

§ 1.11 关于 printf

VOLUME.C 程序如同本书中的大多数例子, 使用 **printf** 库函数来显示文本。为阅读本书的其余部分, 你不必了解 **printf** 的所有细节, 但是如果了解一些基本概念就能更容易明白例子。

printf 函数与 QuickBASIC 的 PRINT USING 语句和 QuickPascal 的 Writeln 过程相似, 它能以不同格式显示(通常在屏幕上)串及数值数据。

传给 **printf** 一个串就能打印出一简单信息:

```
printf( "Hi,Mom!" );
```

该语句打印出

```
Hi,Mom!
```

printf 函数在行尾并不自动地加一新行符, 所以语句:

```
printf( "Le Nozze di Figaro" );
```

```
printf( " by W. A. Mozart" )
```

在同一行上打印如下信息:

Le Nozze di Figaro by W. A. Mozart

为从一新行开始, 使用转义序列\n:

```
printf( "Hi,\nMom!" );
```

该语句在两行上打印两个字:

Hi,

Mom!

printf 中的 **f** 代表格式化。为打印变量及其它项的值, 可以提供给 **printf** 以格式代码。该代码作为第一个变元, 用双引号括起。

下列语句使用 **%x** 代码来以十六进制形式打印整数 553。它传递给 **printf** 两个变元:

```
printf( "%x", 553 );
```

第一个变元("**%x**")含有格式代码, 第二个变元(553)是要格式化输出的项。该行显示为:

229

printf 函数还接受几种其它的格式代码。例如, **VOLUME.C** 程序使用 **%f** 来打印一浮点数。后面章节中的一些程序用 **%d** 来打印整数, 用 **%ld** 打印长整数。

传递给 **printf** 的第一个变元可以包含字符与格式代码的任意组合, 其它变元则为要用 **printf** 进行格式化输出的项。语句:

```
printf( "%d times %d = %d\n", 2, 128, 2 * 128 );
```

打印出:

2 times 128 = 256

printf 函数以从左到右的次序将格式代码与项匹配。在上例中, 第一个 **%d** 与 2 对应, 第二个 **%d** 与 128 对应, 第三个与表达式 **2*128** 对应。

关于 **printf** 和其它 I/O 函数还有非常多的内容, 但对它们的了解可等到你开始学习第十一章“输入及输出”时再进行, 在那里, 我们将详细地描述 I/O。

既然你已经浏览了 C 的全貌, 现在就可以从第二章“函数”开始对 C 程序设计的一些特殊问题加以详细考察了。