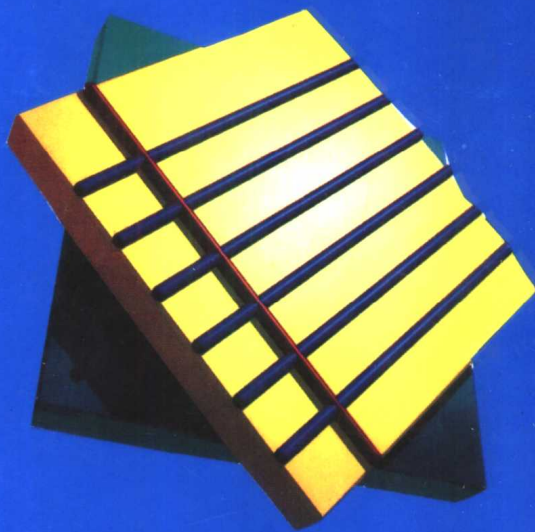


计算机知识普及系列丛书



Quickc、Microdotc

# 高级编程指南

李振格  
王江  
罗克  
汤建平  
编写

学苑出版社

计算机知识普及系列丛书

## *Quick C、Microsoft C* 高级编程指南

李振格	王江	编写
罗克	汤建平	
管叶茂		审校

学苑出版社

1993.

(京)新登字 151 号

### 内 容 提 要

本书的前几章讨论非常有用的数据结构:栈、队列、链接表和树。其后介绍开发通用的应用程序的方法,如语法分析,排序和数据压缩、加密,与 DOS 的接口,利用随机数生成器进行模型仿真,以及图中关键路径,迷宫的求解等。最后在附录中给出了一些 Microsoft C、Quick C 语言和 Quick C 集成环境的参考资料。

欲购本书的用户,请直接与北京 8721 信箱联系,电话 2562329,邮码 100080。

计算机知识普及系列丛书

#### Quick C、Microsoft C 高级编程指南

---

编 写: 李振格 王 江  
      罗 克 汤建平  
审 校: 管叶茂  
责任编辑: 甄国宪  
出版发行: 学苑出版社 邮政编码: 100032  
社 址: 北京市西城区成方街 33 号  
印 刷: 施园印刷厂  
开 本: 787×1092 1/16  
印 张: 38 字 数: 901 千字  
印 数: 1~3000 册  
版 次: 1993 年 12 月北京第 1 版第 1 次  
ISBN7-5077-0821-7/TP·19  
本册定价: 30.00 元

---

学苑版图书印、装错误可随时退换

## 前 言

不管是正式出版，还是非正式出版，用户手册、操作手册、参考手册之类的书都已很多了。每当一个“强大”的软件推出时，都产生一种使用手册；当一个现有软件由于功能或性能更新或提高而出现新版本时，也会产生新手册。用户手册乃软件与用户界面(典型的是中心控制的集成环境)的用法的全面而简单的介绍，参考手册却是软件的增强手段即库函数包中子程序的用法说明，如参数的个数、类型，函数的返回值等。软件的使用法也可以说是语法，它们解释得虽然全面，但略显简单和单调。

对于一个真正的软件开发者来说，掌握一个应用软件(如 AutoCad)或一种语言软件(如 MicroSoft C)只是一种手段，其真正的目的是利用这些软件，设计数据结构去实现一种算法，克服现实生活中的问题。因此追踪软件的版本是徒劳的。最关键的是通晓相对稳定的数据结构和实现算法模型。

《Quick C MicroSoft C 高级编程指南》(上、下册)就是介绍使用 Quick C 和 MicroSoft C 的数据结构和编程技巧的书。

本书的前几章讨论非常有用的数据结构：栈、队列，链接表和树。其后介绍开发通用的应用程序的方法，如语法分析，排序和数据压缩、加密，与 DOS 的接口，利用随机数生成器进行模型仿真，以及图中关键路径，迷宫的求解等。最后在附录中给出了一些 MicroSoft C、Quick C 语言和 Quick C 集成环境的参考资料。

并不是说不需要用户手册、参考手册，相反，它们是基础。因此在阅读本书前，希望对 MicroSoft C 和 Quick C 有一些了解。另外，为了让读者更好地使用 Quick C，我们将很快继续推出《使用 Quick C》。

译者

1991 年 1 月

# 目 录

前 言

第一章 栈与队列.....1

§ 1.1 栈.....1

§ 1.1.1 栈的描述.....2

§ 1.1.2 栈的操作.....2

§ 1.1.3 初始化和考察一个栈.....2

§ 1.1.4 栈的插入与删除.....7

§ 1.1.5 栈的练习.....9

§ 1.2 栈的应用：一个简单的后缀式计算器.....11

§ 1.2.1 中缀记号法.....11

§ 1.2.2 后缀记号法.....12

§ 1.2.3 建立后缀表达式.....13

§ 1.2.4 后缀式的计算.....14

§ 1.2.5 后缀型计算器的实现.....15

§ 1.2.6 中缀型表达式转化为后缀型.....26

§ 1.2.7 建议.....38

§ 1.3 队列.....38

§ 1.3.1 队列处理函数.....40

§ 1.3.2 队列处理函数的练习.....47

§ 1.3.3 任务调度程序.....48

§ 1.3.4 建议.....52

§ 1.4 小结.....52

第二章 递归与链表.....52

§ 2.1 函数调用.....52

§ 2.2 递归

§ 2.2.1 递归计数.....53

§ 2.2.2 斐波纳契数.....58

§ 2.2.3 Hanio 塔.....59

§ 2.2.4 关于递归的几点忠告.....64

§ 2.3 链表.....64

§ 2.3.1 链表处理函数.....64

§ 2.3.2 表处理演示程序.....74

§ 2.3.3	删去链表结点	77
§ 2.3.4	应用举例：词汇测试程序	84
§ 2.3.5	词汇测试器的扩充	89
§ 2.3.6	链表的应用	90
§ 2.4	小结	90
<b>第三章</b>	<b>树</b>	<b>90</b>
§ 3.1	应用举例：邮件的分类	90
§ 3.2	树	93
§ 3.2.1	二叉树的建立	94
§ 3.2.2	二叉树的遍历	97
§ 3.2.3	宽度优先遍历法	100
§ 3.2.4	树的搜索	107
§ 3.2.5	删除树结点	109
§ 3.2.6	二叉树处理函数的演示	114
§ 3.2.7	平衡树	119
§ 3.2.8	AVL 树	121
§ 3.2.9	可改进的例程	129
§ 3.2.10	垂直显示树	129
§ 3.2.11	旋转任意结点	134
§ 3.2.12	练习树处理函数	136
§ 3.2.13	建议	141
§ 3.2.14	小结	141
<b>第四章</b>	<b>排序与查找</b>	<b>141</b>
§ 4.1	算法的有效性	141
§ 4.2	排序	141
§ 4.2.1	选择排序	142
§ 4.2.2	冒泡排序法	145
§ 4.2.3	摇动排序——一种双重的冒泡排序法	147
§ 4.2.4	排序例程的演示	148
§ 4.2.4	插入排序	153
§ 4.2.5	插入排序法的变种。	155
§ 4.2.6	谢尔(Shell)排序	155
§ 4.2.7	快速排序	157
§ 4.2.8	快速排序的缺陷	161
§ 4.2.9	快速排序的改进	162
§ 4.2.10	结构数组的排序	162

§ 4.2.11 链表与树的排序	168
§ 4.3 顺序存取结构：文件	169
§ 4.3.1 简单顺序排序	169
§ 4.3.2 复杂文件的排序	173
§ 4.3.3 建议	175
§ 4.4 查找	175
§ 4.4.1 顺序查找：数组、链表与树	176
§ 4.4.2 折半查找	176
§ 4.4.3 建议	177
§ 4.5 小结	178
第五章 表达式的分析	178
§ 5.1 分析	178
§ 5.1.1 单句语法分析程序	178
§ 5.1.2 简单的数学分析器	190
§ 5.1.3 递归下降分析器	191
§ 5.1.4 操作符优先级规则集	192
§ 5.1.5 变量与函数	192
§ 5.1.6 分析器的语法	192
§ 5.1.7 应用举例	193
§ 5.1.8 语法分析器的实现	196
§ 5.2 利用 QuickC 在内存中编译和运行分析器	212
§ 5.2.1 应用举例	212
§ 5.2.2 扩充语法分析器的功能	228
§ 5.2.3 任务和建议	236
① 新操作符	236
② 显示和保存中间值	236
③ 表达式的循环计算	236
④ 命名公式	236
⑤ 出错处理	237
§ 5.3 小结	237
第六章 数据压缩与加密	237
§ 6.1 压缩策略	237
§ 6.1.1 删去正文文件中的空格	237
§ 6.1.2 恢复压缩文件中的空格	241
§ 6.2 根据字母对进行压缩	243
§ 6.2.1 确定字母对的频率	243

§ 6.2.2	根据字母对频率的压缩算法	253
§ 6.2.3	其它策略	265
§ 6.3	数据加密	266
§ 6.3.1	移位加密法	266
§ 6.3.2	置换加密法	270
§ 6.3.3	解密	272
§ 6.3.4	应用举例	279
§ 6.3.5	密码转换程序	279
§ 6.3.6	问题	297
§ 6.4	任务和建议	299
§ 6.4.1	部分移位加密法	299
§ 6.4.2	关键字加密法	300
§ 6.4.3	利用扩展字符集加密	300
§ 6.4.4	其它解密工具	300
§ 6.4.5	分层式菜单	300
§ 6.4.6	自动猜测	301
§ 6.5	小结	301



# 目 录

第七章 与 DOS 的接口	307
§ 7.1 寄存器	307
§ 7.2 中断	308
§ 7.2.1 使用 DOS 中断的注意事项	308
§ 7.3 利用功能调度器实现中断	308
§ 7.4 使用 BIOS 中断	348
§ 7.5 小结	355
第八章 随机数的产生	356
§ 8.1 随机数	356
§ 8.1.1 随机选择过程的评估	358
§ 8.1.2 用计算机生成伪随机数	359
§ 8.2 产生伪随机数的算法	359
§ 8.2.1 线性同余法	360
§ 8.2.2 算法的实现	361
§ 8.1.3 有关循环长度的研究	363
§ 8.3 二次算法	368
§ 8.4 递推算法	369
§ 8.5 其它算法	371
§ 8.6 随机数发生器的性能评估	372
§ 8.6.1 用 $X^2$ 检验来评估分布	377
§ 8.7 其它随机分布	383
§ 8.7.1 非整型随机数	383
§ 8.7.2 产生非均匀分布	384
§ 8.7.3 随机数发生器的组合	384
§ 8.8 应用举例	391
§ 8.8.1 建议及扩充	400
§ 8.8.2 随机漫游	400
§ 8.9 小结	401
第九章 仿 真	402
§ 9.1 引言	402
§ 9.1.1 仿真的目的	404

§ 9.2 抽样分布	408
§ 9.2.1 蒙蒂·卡洛法	412
§ 9.3 排队问题	412
§ 9.3.1 等待时间	416
§ 9.3.2 建议	418
§ 9.4 制尺子问题	419
§ 9.4.1 尺子的制造	419
§ 9.4.2 产生服从正态分布的随机数	421
§ 9.4.3 标准差的计算	424
§ 9.4.4 产品检验：正态分布的性质	425
§ 9.5 JACKKNIFE 和 BOOTSTRAP 方法	428
§ 9.5.1 Jackknife 方法	428
§ 9.5.2 Bootstrap 方法	434
§ 9.6 产生仿真数据应注意的问题	445
§ 9.6.1 掷两个骰子	449
§ 9.6.2 建议	454
§ 9.7 综合例程	455
§ 9.8 小结	470
第十章 数据分析与统计	471
§ 10.1 基本概念	471
§ 10.1.1 均值与标准偏差	471
§ 10.1.2 均值的另一种表示法：中间值	472
§ 10.1.3 百分位点与标准差	474
§ 10.1.4 用百分位点分析数据	476
§ 10.1.5 如何处理可疑值	478
§ 10.1.6 应用举例	479
§ 10.1.7 建议	495
§ 10.2 统计推断	500
§ 10.2.1 Z 检验	501
§ 10.2.2 比较两个样本	503
§ 10.3 数据拟合	506
§ 10.4 抽样分布的评估	509
§ 10.4.1 建议	512
§ 10.5 小结	512
第十一章 图	513
§ 11.1 基本概述	513

§ 11.2 图的表示	515
§ 11.2.1 基本图函数	516
§ 11.2.2 图的数组表示法	516
§ 11.2.3 图的链表表示法	526
§ 11.3 图的遍历	539
§ 11.3.1 深度优先法	539
§ 11.3.2 广度优先法	548
§ 11.4 搜索最短路径	553
§ 11.5 多结点情况	557
§ 11.6 应用举例	557
§ 11.6.1 用图解决趣味题	558
§ 11.7. 建议	559
§ 11.7.1 “贷郎担”问题	559
§ 11.7.2 结点的命名	560
§ 11.7.3 相邻矩阵例程	560
§ 11.8. 小结	560
<b>附录 A Quick C 简介</b>	<b>561</b>
§ A.1 程序结构	562
§ A.1.1 语句	562
§ A.1.2 标识符与命名规则	563
§ A.1.3 变量说明	563
§ A.2 函数定义与原型	563
§ A.2.1 函数的定义	563
§ A.2.2 函数原型	564
§ A.3 简单数据类型	564
§ A.4 复合数据类型	566
§ A.4.1 数组	566
§ A.4.2 结构	567
§ A.4.3 联合	568
§ A.5 指针	569
§ A.5.1 指针操作符	569
§ A.5.2 指针的算术运算	570
§ A.5.3 函数调用中使用指针传递参数	570
§ A.6 C 语言操作符	571
§ A.6.1 函数调用操作符 ( )	571
§ A.6.2 数组下标运算符 [ ]	572
§ A.6.3 结构成员操作 “.” 与 “->”	572
§ A.6.4 一元运算符	572

§ A.6.5	算术运算符	574
§ A.6.6	移位操作符	575
§ A.6.7	关系运算符	575
§ A.6.8	按位运算符	575
§ A.6.9	逻辑与条件运算符	576
§ A.6.10	赋值运算符 =	576
§ A.6.11	逗号 (,) 运算符	576
§ A.7	控制结构	577
§ A.7.1	if_else 语句	577
§ A.7.2	switch 语句	578
§ A.7.3	while 循环	580
§ A.7.4	do_while 循环	582
§ A.7.5	for 循环	582
§ A.7.6	break 和 continue 语句	583
§ A.8	小结	583
<b>附录 B 在 Quick C 中建立函数库</b>		<b>584</b>
§ B.1	建立 Quick 库	584
§ B.1.1	建立源文件	584
§ B.1.2	编译与连接产生 .qlb 文件	585
§ B.1.3	装入 Quick 库	586
§ B.2	建立独立库	586
§ B.2.1	独立库的使用	591
§ B.3	小结	592
<b>附录 C QuickC 命令</b>		<b>593</b>
§ C.1	调用 QuickC 的命令:	593
§ C.2	常用命令	593
§ C.3	编辑命令	594
§ C.3.1	移动命令:	594
§ C.3.2	选择命令	595
§ C.3.3	插入命令	595
§ C.3.4	删除命令	595
§ C.3.5	其它有用的命令	596
§ C.4	菜单命令	596
§ C.4.1	文件菜单 (ALT-F)	596
§ C.4.2	编辑菜单 (ALT-E)	597
§ C.4.3	查看菜单 (ALT-V)	597

§ C.4.4	搜索菜单 (ALT-S)	598
§ C.4.5	运行菜单 (ALT-R)	599
§ C.4.6	调试菜单 (ALT-D)	600
§ C.5	在 QuickC 环境之外的编译与连接	601
§ C.5.1	存储模式	601
§ C.5.2	预处理程序的控制	602
§ C.6	连接器控制:	602

# 第一章 栈与队列

在本书中你会看到许多有趣的实用程序，这些程序有很大的部分都依赖于很少量的策略与数据结构。本章你将学习两种广泛使用的数据结构。

这些程序大都用于处理由单词、数或者由单词和数的组合所构成的表格和序列。插入和删除是表处理的两个最简单最基本的操作。在本章所考察的那些表中，这两种操作总是在表的某一端即表头或表尾上进行。

可以用两种基本的策略来对这种表或序列进行处理，它们的差别在于元素被加入或被除去的顺序不同。一种策略是总是除去最近加入表中的元素，即最后加入表中的元素最先被删去，这种表叫做 LIFO 表（后进先出表），举例说明 LIFO 表：

- 考古学中的层。即代表特定时期的沉积物。沉积物逐层垒迭，于是最近的沉积物在最顶层。这些层是以后进先出的顺序被挖掘出来的，因为最近的沉积物将会首先被挖掘出来。
- 函数调用。程序在执行一个给定的函数（如 main()）时，调用另一个函数（如 first()），则在执行 first() 时，暂停执行 main()；如果在执行 first() 时，再调用函数 second()，则 first() 又被暂时停止执行，同时将执行 second()。一旦函数 second() 执行完毕，控制就将返回至最近被暂停的函数 first()。假设在下表中的每个函数在执行时都要调用下一个函数，那么这此函数将会以从函数 main() 至函数 fifth() 的顺序被依次暂停执行。但是，这些函数将会以相反的次序被重新激活。即在执行完 sixth() 后，fifth() 将会被重新激活；在执行完 fifth() 后，fourth() 将会被重新激活，并且依次类推。

```
main( )
first( )
second( )
third( )
fourth( )
fifth( )
sixth( )
```

- 自助食堂中堆积起来的盘子和碟子。人们总是取走最顶上的盘子和碟子，并且人们总是将盘子或碟子加到最顶上。

另一种策略是总是除去表中“最老”的元素，即删去最先加入表中元素（也就是表中的第一个元素）。这种策略通常称为“先进先出”。即 FIFO。例如：

- 在银行或自助食堂里的排队。

程序调度者，它将按照收到的控制请求的次序来将控制转交给任务。

- 按照严格的顺序来处理的等待表。

## § 1.1 栈

栈是一种 LIFO 表。可分两部分来描述栈：存储栈元素的空间和存取栈顶元素的方法。

### § 1.1.1 栈的描述

可以用如下的数据结构来描述一个存储有 double 型值的栈。

```
#define MAX_VALS 50 /* maximum number of elements in stack */
struct dbl_stack {
    /* vals is a MAX_VALS element array of double */
    double    vals [ MAX_VALS];
    int       top;
};
```

这一结构包含了栈的两个部分：top 成员总是指明了在栈的数组（即 vals[] 成员）中的下一个可供使用的单元。

类似地，可以使用下面的数据结构来表示一个字符串的栈。

```
#define MAX_VALS 50 /* maximum number of elements in stack */
#define SHORT_STR 15 /* maximum length for a stack entry */
struct stack {
    /* vals is a MAX_VALS element array of strings */
    char    vals [ MAX_VALS] [ SHORT_STR];
    int     top; /* indicates the next stack cell to be filled */
};
```

与 double 型值的栈一样，这个结构也包含了栈的两个部分：（字符串）元素的 vals[] 数组的 top，top 指明了在 vals[] 成员中的下一个可供使用的单元。此时，下一个可供使用的元素本身也是一个数组。因此 top 在实际上是说明了这个字符数组的第一个单元。例如，当 top=10 时，下一个被加入到数组中的字符将会被存储在单元 vals[10][0] 中。

### § 1.1.2 栈的操作

在本章的后几节里，你将编制一些处理字符串的栈的函数。在设计计算器时，将要用到这种表示法所提供的一种灵活性。为此，我们将利用在本章中的第二个栈例子 stack 结构，而不是 dbl\_stack 结构来进行工作。你还可以同样容易地编写出处理其它类型的栈的函数。在读者利用这些例子来进行工作的同时，应该想一下你自己将会怎样来完成这些工作。在后续章节里，你将会使用其它的方式来定义栈元素，并将会对栈函数进行相应的修改。

需要在栈上执行的唯一的破坏性的操作是插入和删除。读者将会发现对一个栈是否为空可是否已满进行检查，以及对诸如栈的大小和栈顶槽中的内容等其它的方面进行考察都是很有用的。此外，一个用于初始化一个栈的通用过程也是很有用的。

### § 1.1.3 初始化和考察一个栈

下述宏和函数实现了栈的初始化、测试栈是否为空栈或满栈，等等。你应输入这些程序，并根据实际情况进行修改。有关 Quick C 的编辑命令参见本书附录 C（或查阅《Quick

C 的使用》第二章)。

```
/* Macros and functions for manipulating stacks.
```

```
File assumes defs.h has been read.
```

```
*/
```

```
/* For all three macros, A should be a pointer to stack */
```

```
#define STACK_EMPTY(A) (!(A)->top) /* is stack empty? */
```

```
#define STACK_FULL(A) ((A)->top == MAX_VALS) /* is stack full? */
```

```
#define STACK_SIZE(A) ((A)->top) /* nr elements in stack */
```

```
struct stack {
```

```
/* vals is a MAX_VALS element array of strings */
```

```
char vals [ MAX_VALS] [ SHORT_STR];
```

```
int top; /* indicates the next stack cell to be filled */
```

```
};
```

```
/* *****
```

```
stack function DECLARATIONS
```

```
***** */
```

```
void disp_stack ( struct stack *);
```

```
void init_stack ( struct stack *);
```

```
void show_stack_top ( struct stack *);
```

```
int stack_height ( struct stack *);
```

```
int stack_is_empty (struct stack *);
```

```
int stack_is_full ( struct stack *);
```

```
/* *****
```

```
stack function DEFINITIONS
```

```
***** */
```

```
/*
```

```
void init_stack ( struct stack *stack_ptr)
```

```
*****
```

```
Initialize each cell in the stack to NULL_STR;
```

```
set top of stack (next element to add) to 0.
```

```
CALLS : strepy ();
```

```
GLOBALS : MAX_VALS
```

```
PARAMETERS :
```

```
struct stack *stack_ptr : pointer to stack being initialized.
```

```
RETURN : <none>
```

```
USAGE : init_stack ( &st);
```

```
*****
```

```
*/
```

```
void init_stack ( struct stack *stack_ptr)
```

```
{
```



```

    /* Index:
    for ( index = 0; index < MAX_VALS; index++)
        strcpy (stack_ptr -> vals [ index] null_str);
    stack_ptr -> top = 0;
}
*/

int stack_is_empty ( struct stack *stack_ptr)
*****

Test whether stack_ptr is empty.
If the_stack.t.p == 0, then stack is empty,
so negating value returns the correct answer.
CALLS :
GLOBALS :
PARAMETERS :
    struct stack *stack_ptr : pointer to stack being checked.
RETURN : TRUE if stack is empty; FALSE otherwise.
USAGE : result = stack_is_empty ( &st);
*****

*/
int stack_is_empty ( struct stack *stack_ptr)
{
    return ( !(stack_ptr -> top)); /* if zero, return true */
}
/*

int stack_is_full ( struct stack *stack_ptr)
*****

Test whether stack_ptr is full.
If stack_ptr -> top == MAX_VALS, then stack is full.
CALLS :
GLOBALS : MAX_VALS
PARAMETERS :
    struct stack *stack_ptr : pointer to stack being checked.
RETURN : TRUE if stack is full; FALSE otherwise.
USAGE : result = stack_is_full ( &st);
*****

*/
int stack_is_full ( struct stack *stack_ptr)
{
    return ( stack_ptr -> top == MAX_VALS); /* if equal, return true */
}

```