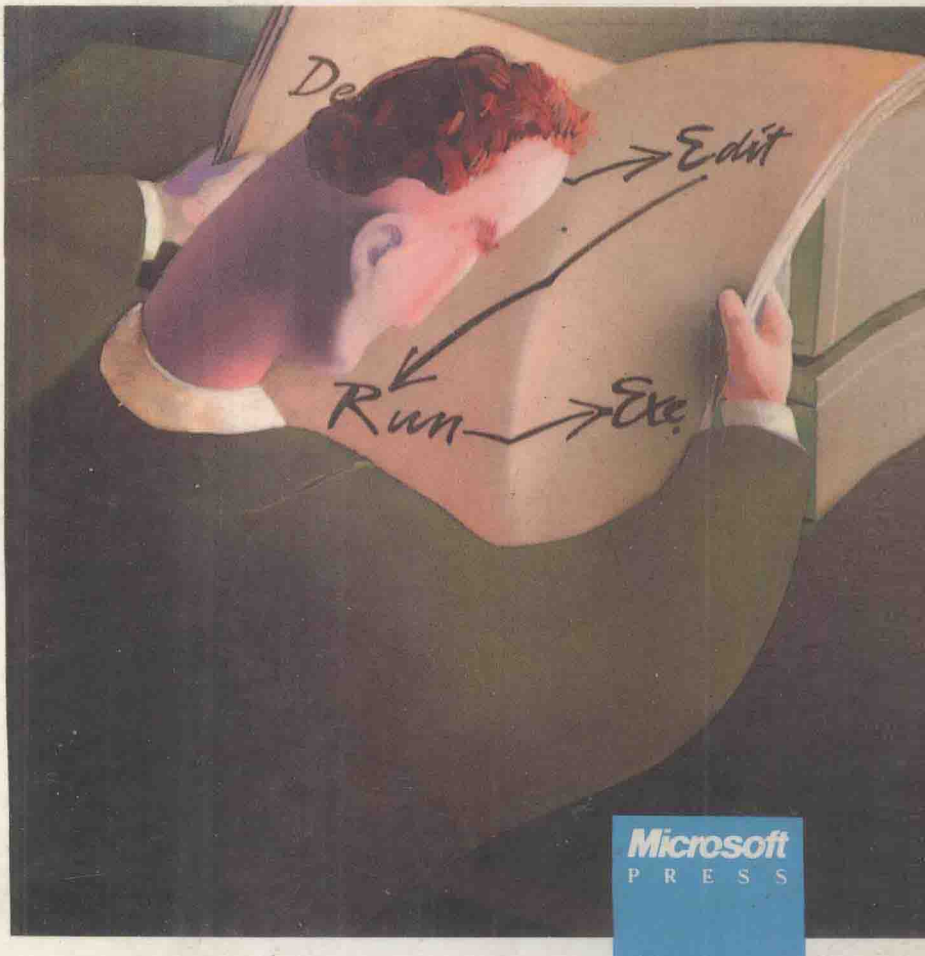
计算机程序设计语言系列丛书

程序员工具箱

An essential library of more than 250 subprograms, functions, and utilities for supercharging QuickBASIC programs



Microsoft PRESS®

JOHN CLARK CRAIG

希望
学苑出版社

计算机程序设计语言系列丛书

Microsoft QuickBasic 4.0
程序员工具箱

John Clark Craig 著
白晓毅 译
熊可宜 审校

学苑出版社

(京)新登字 151 号

内 容 简 介

本书基于微软公司开发的 QuickBasic 4.0。Basic 已经最后发展成一种灵活的、功能齐全的、强有力的程序设计语言。第一章为用户提供了各个指令，以组成一个名为 MINICAL 的完成的有效程序；第二章包含了全部的 QuickBasic 工具箱；第三章讲述的是混合语言工具箱；最后五个附录包含了关于运行工具箱的条件、函数和子程序的交叉引用方面的知识及其他重要的内容。

版 权 声 明

计算机程序设计语言系列丛书

Microsoft QuickBasic 4.0 程序员工具箱

著　　者：John Clark Craig
译　　者：白晓毅
审　　校：熊可宜
责任编辑：甄国宪
出版发行：学苑出版社　　**邮政编码**：100036
社　　址：北京市海淀区万寿路西街 11 号
印　　刷：双青印刷厂
开　　本：787×1092　1/16
印　　张：23.5　　**字　　数**：551 千字
印　　数：1～10000 册
版　　次：1994 年 9 月北京第 1 版第 1 次
ISBN7－5077－0807－1/TP・18
本册定价：38.00 元

目　录

第一章 绪 言

1.1 关于 QuickBasic 与工具箱

基于微软公司开发的 QuickBasic 4.0,BASIC 已经最后发展成一个灵活的、功能齐全的、强有力的程序设计语言。通过翻阅这本书,浏览程序清单,用户可以发现 BASIC 已不再像以前那样。微软公司的 QuickBasic 更易于阅读,学习起来更迅速,并使你有能力很快地创建高级程序,而这则是使用传统的 BASIC 很困难或者不可能做到的事情。

传统 BASIC 与 QuickBasic 之间的关键区别是 QuickBasic 允许结构化程序设计,这是大型程序易于创建与维护的一个重要特性。

1.1.1 结构化程序设计的优点

使用早期版本的 BASIC,一个程序仅仅作为程序行的一个单独的块来编写和执行。不熟练的程序设计者在编写大型程序时,常会无意中产生"细面条式的代码"(即频繁地及不正确使用 GOTO 语句的程序),从而使得程序通常难以跟踪和维护。

QuickBasic 的一个关键特性是它允许用户创建结构化程序的能力——即大型程序由小的、独立的程序模块组成。取代了必须创建并使用大型(并且经常溢出)的单一代码块的工作,QuickBasic 的程序设计者只需创建程序模块,它们由称为子程序和函数的过程按次序组成。这些过程中的每一个都完成一个特定的、完整定义的任务。将注意力集中在单一过程的功能上,程序设计者就不必顾虑程序的其他部分,从而能致力于当前的任务。已经证明,使用这种模块化方式,程序设计者可以更快、更准确地建立复杂的程序。

结构化程序设计的另一个优点是:这些模块和过程能够通过这种方式进行组织和存储,从而在其他程序中每次使用,避免了从一个程序设计项目到另一个项目的重复工作。用互补功能性对模块进行分类,程序员可以很容易地将可用的例程创建成工具箱,在以后,由于程序的主要部分已经编写完毕,因此可以使大型程序设计更快地完成。

在结构化之后,一个模块可以被组织成一个 Quick 程序库,其中的文件在磁盘上按特定格式存放。用户就可以用 QuickBasic 程序安装 Quick 程序库,从而有效地将库中的例程加到其他库上以建成 QuickBasic。

1.1.2 本书中的工具箱

如果用户正在使用(或打算使用)QuickBasic,那么本书就是个有价值的参考。如果用户只是开始,并作为第一门语言来学习 QuickBasic,用户会马上发现这本书对于通过举例学习是有用的。如果用户是一个将 QuickBasic 作为软件开发系统的熟练的专业程序设计者,就会发现本书中的例程是对 QuickBasic 语言有价值的扩展。

第一章为用户提供了逐步的指令,以组成一个名为 MINICAL 的完整的有效程序。特别是

初学的程序设计者会发现这种指导非常有用。第二章包含了全部的 QuickBasic 工具箱，并以解释如何装入和运行工具箱的简要段落开始这一章。最后，五个附录包含了关于运行工具箱的条件、函数和子程序的交叉引用方面的知识以及其他重要的内容。

如果用户是熟练的程序员，可以跳过前面的部分，而进入第二章，开始使用一些工具箱。用户将创建包含函数和子程序的两个模块，并使用它们建立一个 Quick 程序库。一旦用户学会创建自己的程序和 Quick 程序库所需要的步骤，用户就可像使用自己的模块一样来使用本书第二、三章中的模块，建立更快速的程序库。不久用户就会拥有强有力的工具箱，从而能用它来快速地编写程序。

1.2 一个完整的程序——MINICAL.BAS

在这一节里，我们将从勾勒并建立一个完整的可运行的程序开始。我们将会组织这个程序，建立一个扩展 QuickBasic 语言的快速程序库，并建立一个独立的可执行程序。

在开始之前，先大概看一下 QuickBasic 程序设计环境的能力和其中包含的主要概念。这一节中的举例程序由两个相互独立的模块组成，每一个又由几个子程序和函数组成。下面来看一下 QuickBasic 如何处理这些。

1.2.1 模块化源代码编辑

微软 QuickBasic 4.0 的新特性之一是从 QuickBasic 环境中检查和编辑程序的方式。如果用户已经用 QuickBasic 进行过程序设计，也许会注意到由几个子程序和函数组成的程序，不能在一屏上全部显示和编辑。若用户并未用 QuickBasic 进行过程序设计，关于这一点，用户只需知道在当前装入的子程序和函数列表中选择一个作为所要观看和编辑的对象，其余的例程则都隐藏在窗口背后。最初时，这看起来有点奇怪，但在运行了一些程序之后，用户就会体会到这种模块化编辑所带来的作用。

QuickBasic 4.0 的另一个重要的改进是它一次往内存装入多个源代码的能力。这就为创建子程序和函数集合创造了条件，这些子程序和函数按主题存放在不同的源代码文件中，这样几个不同的程序可以各自独立地装入和使用。

关于这些新特色的重要概念，可按下面的方式加以归纳：一个程序可以由一个或多个源代码文件（模块）组成，每个源代码文件可以由一个或多个子程序或函数组成。用户可以同时将这些源文件中的几个装入 QuickBasic 环境中，所有的模块能够一起工作，从而构成一个完整的程序。虽然用户在一次只能显示并编辑一个源文件的一部分，但在编辑一个程序时，可以很容易地从一个部分跳到另一个部分。

1.2.2 建立一个快速程序库

创建新的 QuickBasic 语句和函数，用户可以按照每次启动 QuickBasic 时这些语句和函数即可使用的方式将它们添加到语言中，这样难道不是很好吗？这就是快速程序库为用户所做的！

例如，假设用户在所编写的几乎每个程序中都使用双曲线函数。用户可以在每个源代码文件中创建这些函数，并随着所编写的每个主程序装入到内存中，或者创建一个快速程序库，从

而使得用户在装入 QuickBasic 的同时装入这些函数。要建立一个快速程序库，用户可以简单地装入双曲线函数源代码文件，并从运行菜单中选择适当的菜单选项。

1.2.3 为 MINICAL 创建源代码

下面一步步地学习创立一个完整程序设计项目的过程。启动计算机，接着做下去，以获得最大的好处。MINICAL 程序完成五个功能：加、减、乘、除和开平方。这个程序总体上很简单，但它包括了更大的软件项目的大多数主要成分。

在开始进行编码之前，先来看一下在程序建立后将如何运行。MINICAL 程序用波兰表示法(RPN)进行输入。使用 RPN 可以在很大程度上简化程序设计，这是因为它可以排除编码的必要性，以重新组织在括号中的那些数学命令。

使用 RPN 时，用户先输入数字，接下去是运算符。例如 3 加 4，将输入：

```
3 4+
```

(在这一节的后面，我们将会介绍这些数字实际上是如何输入的)。要完成 1 加 2，然后乘以 3，则输入：

```
1 2 +3*
```

MINICAL 使用双精度，因此用户可以输入整数或浮点数值。结果使用 16 位数字显示。例如，1.2 除以 −3.45，则输入：

```
1.2 −3.45/
```

显示结果为：

```
Result. . . −. 3478260869565217
```

注意：因为计算机键盘没有×和÷，所以用星号(*)代表乘，用正向斜杠(/)表示除。

顺便提一下，MINICAL 在完成计算时，使用一个称为栈的结构存放数字和运算符(从技术上讲，在 MINICAL 中使用的这一结构，只是模仿传统的堆栈。为了讨论方便，它可以被看作是一个堆栈)。堆栈是内存位置的有序系列，这些内存被预留出，每个单元存放一个数字。在这种情况下，数字和运算符由用户提供。由于堆栈的存在，RPN 主要用来指定数字和运算符——RPN 句法是一种基于栈运算的思想。另一种可选择的方式是分析，它包括阅读用户输入的等式，重新组织，并选择可分离的元素，并对它们进行操作。

MINICAL 中的堆栈可以存放 20 个值以及相关的运算符。用户可以像上面解释的那样输入数字，或者先输入所有的数字，然后是运算符。例如，计算从 1 到 5 数字的和，可以按下面两个命令行之一的方式进行输入：

```
1 2 + 3 + 4 + 5 +
1 2 3 4 5 + + + +
```

1.2.4 MINICAL 模块

这个项目包括两个部分：MINIATH 模块和 MINICAL 模块。下面从 MINIMATH 开始。

如果还未进行启动，则在系统提示符下打入 QB，并按回车以启动 QuickBasic。在下面的标题块中输入：

```
' ************************************************
' **  Name:          MINIMATH                    **
' **  Type:          Toolbox                     **
' **  Module:        MINIMATH.BAS                **
' **  Language:      Microsoft QuickBASIC 4.00   **
' ************************************************
'
' Collection of math subprograms for the MINICAL
' program.
```

在这里很方便地告诉 QuickBasic 这个模块的名字。按下 File 菜单，选择 Save As。打入文件名 MINIMATH，并按回车，或将鼠标指针移到 OK 框，并单击左按钮。这个文件就被存到磁盘上，并且用户已准备好继续键入这个程序。如果用户略去了.BAS 扩展名，如前所做的那样，QuickBasic 会自动为用户添上。

第一个模块由完成数学功能的五个子程序组成。首先建立 Add 子程序，拉下 Edit 菜单，并选择 New SUB。在用户被询问到子程序名字时，回答 Add。QuickBasic 就会创建新的子程序的第一行和最后一行：

```
SUB Add
END SUB
```

注意 QuickBasic 也调整编辑窗口，只显示 Add 子程序，使得用户可以集中精力到该子程序上(在后面用户将会随着程序设计项目的变大，愈来愈欣赏这一新特色)。下一步是在子程序的第一行之前，添加注释信息，然后在 QuickBasic 显示的两行之间加入子程序的"内脏"。

先开始添加注释。将光标移到子程序第一行的第一个字符处，然后按回车。这时就会出现一个对话框，其中有信息"Blank lines not allowed before SUB/FUNCTION line. Is remark OK?"既然注释是用户所要的，则选择 OK。QuickBasic 就会为注释插入一个以'字符开头的空行。在打入第一个注释行(如下面所示)之后按回车，对话框会再次出现。单击 OK，然后打入下面的注释行。

然后使用下面所示各行建立 Add 子程序。注意 SUB Add 和 END SUB 行的位置，并且要注意用户需要在 SUB Add 行后添加条目。另外还要注意一些行是排列成锯齿状的。虽然这种锯齿形状不是必要的，但是将下一级的行排成锯齿状是一种良好的程序设计风格，这样会使程序易于阅读，而且行与行之间的关系一目了然。完成这些后，Add 子程序如下所示：

```
' ************************************************
' **  Name:          Add                         **
' **  Type:          Subprogram                  **
' **  Module:        MINIMATH.BAS                **
' **  Language:      Microsoft QuickBASIC 4.00   **
' ************************************************
'
' Performs addition for the MINICAL program.
'
  SUB Add (stack#(), ptr%) STATIC
      ptr% = ptr% - 1
```

```
    IF ptr% THEN
        stack#(ptr%) = stack#(ptr%) + stack#(ptr% + 1)
    END IF
END SUB
```

用户可以在任何时候将所做的程序存储起来(经常将编写的程序备份下来,以防止由于电源故障或其他原因而使程序丢失是个好主意)。这时可拉下 File 菜单,并选择 Save all 来实现这一功能。

到目前为止,一切进展很顺利。用户可以用同样的方法创建并编辑其他四个数学子程序。它们分别如下所示。

这有一个提示,可以提高这个过程的速度。注意到五个子程序中的每一个开始的注解行几乎是相同的。因此,可以取代每个程序重复打印的工作,而代之以拷贝用户为 Add 子程序所打印的注解行,并将它们放到其他子程序中。下面就来完成这一工作。

首先建立新的子程序。对于减法子程序,从 Edit 菜单中选择 New SUB,输入名字 Subtract,并按回车。在减法子程序的两行显示出来之后,重复为乘、除及开平方子程序进行 New SUB 过程。要保证用户所打入的子程序名字以大写字母开头,并且在子程序名字 SquareRoot 中不允许有空格。

然后对 Add 子程序中的注解行进行拷贝。拉下 View 菜单,并选择 SUBS。QuickBasic 会显示一个用户输入的所有子程序名字的列名。在这里将显示出 QuickBasic 的有力功能:使用 SUBS 命令,用户可以跳到任何一个子程序进行编辑,而这只需简单地选择所要的子程序即可实现。用户可以通过用鼠标在所要的子程序名字上双击或使用光标移动键和回车键达到这一选择的目的。在这里选择 Add。

为了拷贝注解行,将光标移到第一个注解行的第一个字符处,然后使用键盘按着任何一个 Shift 键,并按向下箭头键,直到所有的注解行都用高亮度显示;或者使用鼠标,将其箭头移到光标的位置,按下左按钮,并向下拖动鼠标,直到所有的注解行都用高亮度显示。最后,从 Edit 菜单中选择 Copy,将高亮度行拷贝到 Clipboard(剪贴板)。

接下来,将这些行拷贝到其他四个子程序中。从 View 菜单中选择 SUBS,再次显示子程序名清单。选择 Subtract,将光标移到第一行的第一个字符处,拉下 Edit 菜单,选择 Paste(粘贴)。这里从 Add 子程序中拷贝的行就会显示出来(如果这些行未能显示,或者用户得到一个对话框,提示用户空白行不能在一个子程序前显示,就返回到 Add 子程序,重复 Edit－Copy 过程。记住只拷贝以' 字符开头的注解行)。

用同样的方法选择其他的子程序名,并重复 Edit－Paste 过程。用户不必重复 Copy 操作,这是因为拷贝到 Clipboard 上的条目将一直保存到有其他内容拷贝到 Clipboard 上或用户退出 QuickBasic 为止。在进行操作时,返回到每个子程序,就像对 Add 子程序的操作一样编辑注解行,并输入程序行。在每个子程序编译完后,要确保从 File 菜单中选择 Save All。另外,还要保证返回并浏览用户所做的工作——使用 View 菜单上的 SUBS 选项分别查看每个子程序的功能,使得这一重要任务变得更容易,而且用户程序更易于阅读。工作的结果如前面所示。

注意:不要忘记编辑注解行!记住用户所剪贴的注解行是 Add 子程序的注解行。用户必须修改在每个注解块中的名字和块下面行中的信息,使它们唯一地与子程序相对应。

下面是减法子程序:

```
' ************************************************
' **  Name:          Subtract                   **
' **  Type:          Subprogram                 **
' **  Module:        MINIMATH.BAS               **
' **  Language:      Microsoft QuickBASIC 4.00  **
' ************************************************
'
' Performs subtraction for the MINICAL program.
'
  SUB Subtract (stack#(), ptr%) STATIC
      ptr% = ptr% - 1
      IF ptr% THEN
          stack#(ptr%) = stack#(ptr%) - stack#(ptr% + 1)
      END IF
  END SUB
```

下面是乘法子程序：

```
' ************************************************
' **  Name:          Multiply                   **
' **  Type:          Subprogram                 **
' **  Module:        MINIMATH.BAS               **
' **  Language:      Microsoft QuickBASIC 4.00  **
' ************************************************
'
' Performs multiplication for the MINICAL program.
'
  SUB Multiply (stack#(), ptr%) STATIC
      ptr% = ptr% - 1
      IF ptr% THEN
          stack#(ptr%) = stack#(ptr%) * stack#(ptr% + 1)
      END IF
  END SUB
```

```
' ************************************************
' **  Name:          Divide                     **
' **  Type:          Subprogram                 **
' **  Module:        MINIMATH.BAS               **
' **  Language:      Microsoft QuickBASIC 4.00  **
' ************************************************
'
' Performs division for the MINICAL program.
'
  SUB Divide (stack#(), ptr%) STATIC
      ptr% = ptr% - 1
      IF ptr% THEN
          stack#(ptr%) = stack#(ptr%) / stack#(ptr% + 1)
      END IF
  END SUB
```

下面是平方根子程序：

```
' ***********************************************
' **  Name:          SquareRoot                 **
' **  Type:          Subprogram                 **
' **  Module:        MINIMATH.BAS               **
' **  Language:      Microsoft QuickBASIC 4.00  **
' ***********************************************
'
' Determines square root for the MINICAL program.
'
  SUB SquareRoot (stack#(), ptr%) STATIC
      stack#(ptr%) = SQR(stack#(ptr%))
  END SUB
```

到此就完成 MINICAL 程序的第一个部分。用户所创建并以 MINICATH 存储的子程序构成了 MINICAL 的核心，因为它们完成了实际的计算。下面一部分包括了创建 MINICAL 本身。MINICAL 完成附加工作——取出用户要进行计算的数字，将它们传递到 MINIATH 中相应的子程序里，并显示计算结果。

1.2.5 MINICAL 模块

从现在开始，用户可以使用两种方法中的一种进行操作。因此 MINICAL 和 MINIMATH 都比较小，用户可以通过创建名为 MINICAL 的模块，直接在内存中建立完整的 MINICAL 程序（要做到这些，用户可从 File 菜单中选择 Create File，接受 Module 的缺省选择，并选择 OK）。或者将 MINIMATH 转变成一个快速程序库，装入 QuickBasic 系统，从而菜单使它变成语言的延伸。在这里使用第二种方式来看一下使用这种高级 QuickBasic 特性有多容易。

要建立一个快速程序库，从 Run 菜单中选择 Make Library。用户要被询问库的名字。MINIMATH 是比较合适的，因为 QuickBasic 自动为快速程序库添加缺省扩展名. QLB，为它建立的一般库添加. LIB 扩展名。在打入 MINIMATH 后，选择 MakeLibrary（在此用户可忽略对话框中的其他选择）。完成这一工作后，QuickBasic 就在当前目录中创建了两个新文件：MINIMATH. QLB 和 MINIMATH. LIB。

现在退出 QuickBasic 以便能重新启动，并装入新的快速程序库。要完成这一功能，可拉下 File 菜单，并选择 Exit。然后通过输入：

```
QB/L MINIMATH
```

在系统提示符下启动 QuickBasic。用户看不到有什么明显的不同，但实际上已发生某些令人兴奋的事情！用户的 QuickBasic 已被扩展了。现在它远远超过以前的能力。MINIMATH 中的子程序成为 QuickBasic 语言的一部分，可以像许多其他的 QuickBasic 关键字一样使用。实际上，因为用户可以在调用子程序时使用 CALL 关键字，这些新的子程序就像 QuickBasic 中新的关键字一样出现。

继续程序的剩余工作，使得用户可以试用这一新的扩展了的 QuickBasic。要保证像以前描述的那样装入 QuickBasic，并将 MINIMATH 快速程序库作为系统的一部分。然后打入下面的

程序:MINICAL.BAS。

```
' ************************************************
' **  Name:          MINICAL                     **
' **  Type:          Program                     **
' **  Module:        MINICAL.BAS                 **
' **  Language:      Microsoft QuickBASIC 4.00   **
' ************************************************

' Functions
DECLARE FUNCTION NextParameter$ (cmd$)

' Subprograms
DECLARE SUB Process (cmd$, stack#(), ptr%)
DECLARE SUB DisplayStack (stack#(), ptr%)
DECLARE SUB Add (stack#(), ptr%)
DECLARE SUB Subtract (stack#(), ptr%)
DECLARE SUB Multiply (stack#(), ptr%)
DECLARE SUB Divide (stack#(), ptr%)
DECLARE SUB SquareRoot (stack#(), ptr%)

' Get the command line
cmd$ = COMMAND$

' Create a pseudo stack
DIM stack#(1 TO 20)
ptr% = 0

' Process each part of the command line
DO UNTIL cmd$ = ""
    parm$ = NextParameter$(cmd$)
    Process parm$, stack#(), ptr%
    IF ptr% < 1 THEN
        PRINT "Not enough stack values"
        SYSTEM
    END IF
LOOP

' Display results
DisplayStack stack#(), ptr%

' All done
END
```

这是MINICAL程序的主要部分,所有操作都从这里开始。注意DO—LOOP结构中的前两行,即parm$=Next Parameter$(cmd$)和Process parm$,Stack#(),ptr%。第一行调用了用户定义的名为Next Parameter$的函数,第二行调用了用户定义的名为Process的子程序(它们并不是你所定义的。它们是任务表中的下一步所完成的)。注意,并没用关键字CALL去调用Process子程序。若想要的话,用户可以使用CALL,但是没必要再用。由于QuickBasic处理子程序的方式,用户不久后创建的Process子程序更像QuickBasic系统的一部分,而不像程序的部分,因为用户不能在程序的这部分显示在屏幕上时对它列出清单并修改。用户的创造,可用于对付下面更高层次的程序复杂性。

一旦用户输入了主程序行,就要再次将这个模块存放到磁盘上。从File菜单中选择Save

As，并输入文件名 MINICAL。

在试运行这个程序之前，用户还有几段代码要完成。要创建这个程序的一个函数，就从 Edit 菜单中选择 New Function。打入函数名 Next Parameter $，并按回车键。实际上，创建并编辑函数与创建并编辑子程序没什么区别。实事上函数和子程序之间唯一的主要区别是：函数要返回一个用于计算或为 QuickBasic 变量赋值的一个数值。子程序通过传递的变量返回值。使用与创建 MINIMATH 中子程序同样的步骤来创建函数 Next parameter $：

```
' ************************************************
' **  Name:          NextParameter$              **
' **  Type:          Function                    **
' **  Module:        MINICAL.BAS                 **
' **  Language:      Microsoft QuickBASIC 4.00   **
' ************************************************
'
' Extracts parameters from the front of the
' command line. Parameters are groups of any
' characters separated by spaces.
'
FUNCTION NextParameter$ (cmd$) STATIC
    parm$ = ""
    DO WHILE LEFT$(cmd$, 1) <> " " AND cmd$ <> ""
        parm$ = parm$ + LEFT$(cmd$, 1)
        cmd$ = MID$(cmd$, 2)
    LOOP
    DO WHILE LEFT$(cmd$, 1) = " " AND cmd$ <> ""
        cmd$ = MID$(cmd$, 2)
    LOOP
    NextParameter$ = parm$
END FUNCTION
```

现在创建并编辑下面两个作为 MINICAL 模块一部分的子程序。

创建子程序 Display Stack：

```
' ************************************************
' **  Name:          DisplayStack                **
' **  Type:          Subprogram                  **
' **  Module:        MINIGAL.BAS                 **
' **  Language:      Microsoft QuickBASIC 4.00   **
' ************************************************
'
' Displays anything left on the stack when MINICAL
' finishes processing the command line.

SUB DisplayStack (stack#(), ptr%) STATIC
    PRINT
    IF ptr% > 1 THEN
        PRINT "Stack... ";
    ELSE
        PRINT "Result... ";
    END IF
    FOR i% = 1 TO ptr%
```

```
        PRINT stack#(i%),
    NEXT i%
    PRINT
END SUB
```

接着创建子程序 Process：

```
' ***********************************************
' **  Name:          Process                   **
' **  Type:          Subprogram                **
' **  Module:        MINICAL.BAS               **
' **  Language:      Microsoft QuickBASIC 4.00 **
' ***********************************************
'
' Processes each command parameter for the MINICAL
' program.
'
SUB Process (parm$, stack#(), ptr%) STATIC
    SELECT CASE parm$
    CASE "+"
        Add stack#(), ptr%
    CASE "-"
        Subtract stack#(), ptr%
    CASE "*"
        Multiply stack#(), ptr%
    CASE "/"
        Divide stack#(), ptr%
    CASE "SQR"
        SquareRoot stack#(), ptr%
    CASE ELSE
        ptr% = ptr% + 1
        stack#(ptr%) = VAL(parm$)
    END SELECT
END SUB
```

一定要选择 File 菜单中的 Save All 来将所做的工作保存到磁盘上。

完成了这些，然而还有最后一个细节。这个程序读入注解行，并假定数字和运算符是在系统提示符下随着程序的名字打入的。幸运的是，QuickBasic 提供了一种途径，可以让用户在命令行中打入，即使用户目前正从 QuickBasic 系统运行内存中的程序。从 Run 菜单中选择 Modify COMMAND$。要求用户输入一个新的命令行。先试着输入：

3 4 +

现在一切都准备好了，开始运行这个程序。从 Run 菜单中选择 Start。如果正常的话，可看到如下结果：

Result...7

如果不正常，用户可能会发现 QuickBasic 反回一个错误信息，这一错误可能是间接的打印错误。若是这样，重复检查输入，如果问题在 MINIMATH 模块中，则重建程序库。

一旦使得程序运行起来，用户就可花些时间去尝试不同的命令行参数。看一下，如果栈中有几个数字而没有足够的运算符使得栈归约到最后结果，会发生什么现象。例如，试着输入：

3 4 5 6 7 + *

同样也看一下，如果栈上没有足够的数字会怎么样。例如输入：

34 + *。

1.2.6 作为一个可执行程序(.EXE)进行编译和运行

最后,看一下用户如何通过从 Run 菜单中选择 Make EXE File 去创建可从 MS-DOS 中运行的程序。用户可以创建两类.EXE 文件。第一类会产生一个较小的 MINICAL.EXE 文件,但它要求在运行时访问名为 BRUN40.EXE 的 QuickBasic 文件。第二类会产生较大的 MINICAL.EXE 文件,但它可独立存在。当用户从 Run 菜单上选择 Make EXE File 时,系统会提示用户选择所要创建的.EXE 文件的类型。两种方法都试试。看一下文件的大小,并注意 BRUN40.EXE 文件必须在当前目录下,或在通过 MS-DOS 的 PATH 设置定义的某处可访问才行(用户的 QuickBasic 手册更详细地讨论了这一问题)。

另一种方式,从系统提示符下运行 MINICAL.EXE 程序,按 COMMAND$ 所期望的方式使用了命令行。例如,17 减 5,在系统提示符下输入:

```
MINICAL 17 5 -
```

在建立 MINICAL 时,用户会发现创建自己的工具箱或编辑已存在的工具箱有多容易。现在转向本书的第二部分。一定要阅读第一节,在那里解释了如何使用 QuickBasic 工具箱。然后选择一个所感兴趣的工具箱会有很大乐趣。

第二章 QuickBasic 的工具箱及程序

2.1 使用 QuickBasic 工具箱

第二章中的工具箱覆盖了很大范围的论题，并按字母顺序排列。每一个都设计成可由用户编写的应用软件来装入和调用。在开始每个工具箱之前，用户可运行演示模块来显示一下该工具箱中的例程，也可忽略演示模块，并如其中所示，使用这些例程，或重新组织它们以适合用户的应用需要。

工具箱和实用程序并不需要任何先验知识，因此用户可按任意次序加以运行。每个至少尝试一次，在运行时，检查一下代码。用户会发现在大多数列表中的独特技术和程序设计概念。用户还会发现这些工具箱例程连同用户自己创建的例程在将来的程序设计项目中很有用。

许多实用程序都使用命令行输入法，或从 QuickBasic 中的 COMMAND$ 变量来向程序传递数值或变量。其余的(从不同程序中使用工具箱的那些)可能要求一个相应的.MAK 文件。一些程序需要彩色或图形功能，其余的可能需要鼠标。查看每个清单开始的注解或附录 A，以便为每个工具箱和实用程序确定环境和运行条件。

2.1.1 特殊条件

下面的工具箱要求在 QuickBasic 中将 MIXED.QLB 快速程序库装入内存中：BIOSCALL、CDEMO1、CDEMO2、COLORS、DOSCALLS、FILEINFO、MOUSGCRS、MOUSSUBS、MOUSTCRS、QBTREE、STDOUT 和 WINDOWS。MIXED.QLB 由一些按汇编语言和微软 QuickC 编写的子程序和函数组成(MIXED.QLB 的汇编语言和 C 的源程序清单在第三章)。

尽管并不是本书中的所有工具箱都需要有 MIXED.QLB，但在用户每次启动 QuickBasic 以使用本书中的工具箱时，都装入它仍是个好的主意。那些不需要 MIXED.QLB 的工具箱，若装入了它也不会受到影响。

下面是创建 MIXED.QLB 的指令，用微软 Quick C 和汇编语言编写，以显示其他语言如何使用 QuickBasic。详细地解释混合语言程序设计概念，已超出了本书的范围，所以这里只简单地给出创建 MIXED.QLB 的步骤。在本书第三章中，用户有机会试一些混合语言程序的例子。

2.1.2 创建 MIXDE.QLB

如果用户有微软 QuickC，第一步就是为 CTOOLS1.C 和 CTOOLS2.C 编译一个目标代码文件。

用户可在后面 3.2，3.3 中找到 CTOOLS1.C 和 CTOOLS2.C。一旦用户有了这些文件，就在系统提示符下输入下面命令，编译 CTOOLS1.C 和 CTOOLS2.C 的源代码文件，以创建目