在C语言中使用dBASE

——Code Base Ⅳ使用手册

尹勤 美莹 编译

- 实现dBASE的全部数据操作
- 具有窗口和菜单
- 提供全套源程序
- 支持DOS、Windows、OS/2、Unix和Xenix
- 支持Turbo C Microsoft C Quick C
  Zortech C++ Watcom C-386

微宏电脑软件研究所 m&M

目 录

第0章 引 言

Code Base 是一个用于数据库和屏幕管理的 C 语言库，Code Base 与 dBASE 的兼容性使它可以使用任何其它 dBASE 兼容产品。

Code Base 使用与 dBASE 同名约定，因而易于掌握。任何人，只要熟悉 C 程序设计语言并了解一些数据库原理，学习 Code Base 都不会感到困难。

所有学习 Code Base 的用户都应了解这部分。快速参考部分将告诉你哪些例行程序是最急需的。"例行程序说明"这一章中各例行程序前的简介往往是很重要的，建议在使用一个例行程序前先阅读相应的简介。

打算在多用户环境下使用 Code Base 的用户需要阅读"有关多用户的考虑"这一章。

重点是 'Database(数据库)'，'Field(段)'，'index File(索引文件)'，'windowing(窗口)'，'Menuning(菜单)' 和 'get(获取)'。这些例行程序对许多人都是最重要的。

'Browse' 例行程序教你建立用户浏览和编辑屏幕，而这又需要首先学会 'get' 例行程序和一些 'windowing' 例行程序。

对关系和过滤感兴趣的读者不妨参考"扩充例程"。

0.1 基本术语

下面是一些在 Code Base 手册中使用的术语的定义。

数据库

一个数据库是一个数据文件。例如，一个通讯录数据库包含人的姓名和地址。

记录

数据库信息由记录构成。在通讯录数据库中，每个记录包含一 个人的信息。

记录缓冲器

对每个数据库，Code Base 专门为一个记录的信息分配一个存贮区。每个这样的存贮区称为一个"记录缓冲器"。许多高级 Code Base 例程，例如 'd4go' 自动使用这些存贮缓冲器。指向记录缓冲器的指针可用"f4record"例程得到。下节 Code Base 内部代码将描述记录缓冲器的格式。

记录号。

数据库中的每个记录都被赋予一个从 1 开始的顺序记录号。有一个当前记录号对应于记录缓冲器中的记录。当前记录号可用"d4recno"例程得到。

字段

记录由字段组成，一个数据库字段描述每个数据库记录的一个信息类。通讯录数据库中的字段可以是：姓、住址、所在城市和电话号码。例如：LAST__NAME，ADDRESS，CITY，和 PHONE__NUM 是对应的段名。

字段可以是字符型、数字型、浮点数、逻辑型、日期或备注型。字段的类型是很重要的，因为它确定了字段所存信息的种类，字段名和字段类型在数据库生成时就被确定了。

索引文件

在不同的时刻可能需要按不同的次序查询数据库，例如通讯录数据库有时要求按LAST_NAME(姓)字段排序，有时又要求按ADDRESS(地址)字段排序。如果每提出一种新的要求都重新排一次序这时大型数据库需要很长时间。

因此建立索引文件用于永久地保存排好序的信息。一个索引文件为一个数据库保存一种排序结果。索引文件具有B+树结构，意味着能够快速有效地对排序信息定位。

索引文件的表达式

索引文件还含有表明数据按什么要求排序的信息，称为表达式。例如LAST_NAME是一个表达式，表明该索引文件保存的排序结果是按LAST_NAME排序的。

引用号

Code Base几乎处处都要用到引用号，每个Code Base使用的数据库、索引文件、字段、入口、窗口、菜单项都有一个对应的引用号。引用号是许多Code Base例行程序的参数。

0.2 数据库基本概念

下面给出一些本用户手册涉及的基本概念，其余假定读者已经熟悉。

使用Code Base可能要保存和存取多达几百兆字节的信息，这些信息以数据库文件的形式保存在磁盘上。每一时刻，全部有效信息中只有一部分被处理。

可能有几个数据库文件同时被打开，但在某一时刻仅有一个被选中。数据库例程只作用于这个被选文件。打开、关闭数据库要占用许多机时，而在打开的数据库中选择则几乎是立即响应的。

每一个打开的数据库都有一个称为记录缓冲器的内存区。记录缓冲器的精确格式在附录中说明，即在Code Base内部代码一节中描述。许多高级数据库管理例程都使用记录缓冲器，例如，例程"d4go"把一个数据库记录从磁盘读到记录缓冲器，类似地，例程"d4write"和"d4append"直接把数据库缓冲器的内容写到数据库文件中去。

欲存取或修改记录缓冲器，使用Field(段)例程。用Field例程修改记录缓冲器时，产生的改变自动写磁盘上去。

每个数据库可能有一组打开的索引文件，每个索引文件对应于一个特定的数据库。数据库被修改后，相关的索引文件自动被修改。每个数据库都可以有自己的"被选"索引文件。高级数据库管理例程，例如'd4seek'和'd4skip'使用属于当前被选数据库的当前被选索引文件。低级索引文件例程，例如'i4seek'和'i4skip'一般不直接使用。

Memo(备注)文件用于存贮可变长度的正文。为提高效率，每个字段在数据库记录中占固定空间。一个字段在一个数据库记录中占10个字节，在一个Memo文件中占有可变的存贮空间。除了文件扩展名从DBF变为DBT，memo文件与相应的数据库文件同名。一个特定的数据库文件的所有字段使用同一个memo文件。

0.3 启动

第一次启动Code Base需要完成以下步骤：

1. 准备一片工作盘。
2. 细心核对README文件，以便得到最新信息。
3. 阅读"编译器说明信息"。

4. 安装 Code Base。
5. 运行"d4learn"以检验 Code Base 例程。结合一些程序实例阅读文件。
6. 阅读"生成用户 Code Base 应用程序"。
7. 用 Code Base 编程，如有困难，可参考"常见问题"。

0.4 编译器说明信息

目前，Code Base 直接支持 Turbo C、Quick C、Microsoft C、Zortech C++，Watcom C386 和 Sco Unix 或 Xenix 操作系统。如果使用其它的编译器或操作系统，由于 Code Base 提供完整的源代码和测试代码，可以自己建库，经调试使其正常工作。测试代码在磁盘目录"\TEST"下，有关文件在"\TEST\TEST. DOC"下提供。

同时提供的还有两个予先建立的库 T4. LIB 和 M4. LIB，分别用于 Turbo C 和 Microsoft C。M4. LIB 还可以用于 Quick C 2.0 或更高版本。这些库都是用大的存贮模型建立的，使用 Code Base 屏幕管理，dBASENDX 索引文件和 DOS 操作系统进行浮点仿真。如果你有一个不同的编译程序、操作系统或需要不同配置，请查阅附录中建库部分。

在 Turbo C 环境下使用 Code Base

为了在 Turbo C 交互环境下使 Code Base 库，需要一个"project"文件。"D4LEARN. PRJ"是一个 project 样本文件，用于生成程序' D4LEARN. EXE'。这一样本在磁盘目录' \EXAMPLES' 下，包含两行信息：

```
D4LEARN. C
T4. LTR
```

Turbo C 栈容量的缺省值是 3000 字节。如果索引文件很大，为保险起见，应在用户程序顶部增加一行代码以把栈扩充到最大：

```
extern unsigned _stklen = 10000 ;
```

另外，在磁盘目录"\TC"下有一个"TURBOC. CFG"文件，该文件说明"TCC"命令行编译程序的选择项，包含如下信息：

```
-LC:\TC\LIB -IC:\TC\INCLUDE -ml -c -DTURBO -N
```

这行信息描述了库目录，include 目录，大的存贮模型、说明只编译，定义了"TURBO"开关。"TURBO"开关用于说明将使用任何 Turbo C 专用源代码。Code Base 源码在"#ifdef"和"#ifndef"予处理程序中使用"TURBO"开关指向编译程序。

在磁盘目录"\TC"下有一个批文件"C4TC. BAT"，这个文件用于编译和连接小的 Code Base 应用程序。

在 Quick C 环境下使用 Code Base

为在 Quick C 交互环境下使用 Code Base 库"M4LIB"，需要一个"MAKE"文件。通过 Quick C 菜单选择项，增加两个文件"D4LEARN. C"和"M4. LIB"，这两个文件帮助建立 make 文件并编译和连接"D4LEARN. EXE"程序。

必须说明大的存贮模型，分配额外的栈空间，栈空间约为 10K 最为保险。在 Quick C 交互环境下，有菜单选择项。

Quick C 命令行编译程序"QCL. EXE"使用方法与"CL. EXE"相同、"CL. EXE"是 Microsoft

C 编译程序，请参考下节“ 在 Microsoft C 环境下使用 Code Base”。

在 Microsoft C 环境下使用 Code Base

运行命令行编译程序“CL. EXE”进行编译，要求说明大的存贮模型和额外的栈空间。例子：

```
CL /AL D4LEARN.C /link M4.LIB /STACK:10000
```

有关编译和连接选择项的详细解释，请参考《编译程序用户指南》。在磁盘目录“\MSC”下，有一个批文件“C4MSC.BAT”，用于编译和连接小的 Code Base 应用程序。

在 Zortech C++和 Watcom C 386 环境下使用 Code Base

0.5 安装 Code Base

使用 Code Base，仅需要拷贝专用的库文件，头文件和配置信息即可。

A. 从三个予建库中拷贝一个适用的。如果没有适用的可供选择，参考附录的建库部分。

Turbo C 命令格式：

```
COPY A:\TC\T4.LIB  C:TC\LIB
```

其中“COYP”命令把与 Turbo C 兼容的 Code Base 库从 A 盘传送到 C 盘目录“\TC\LIB”下。

Microsoft C (Quick C)命令格式：

```
COPY A:\MSC\M4.LIB  C:\MSC
```

该命令把与 Microsoft C 兼容的 Code Base 库从 Code Base 盘目录“MSC”下拷贝到“C:\MSC”下，如果 Microsoft C 在 OS/2 环境下使用，请参考附录的建库部分。

B. 从 Code Base 盘目录“\H”下拷贝头文件。注意 Turbo C 头文件与 Microsoft C 和 Quick C 的头文件相同。

命令格式：

```
COPY A:\H\*.*  C:\TC\INCLUDE
```

上述命令把所有的 Code Base 头文件盘拷贝到 Turbo C 的“Include”目录下。这些头文件是使用 Code Base 必不可少的。

C. 如果打算使用 Code Base 源文件，需要拷贝 Code Base 盘目录“\SOURCE”的内容。

命令格式

```
COPY A:\SOURCE\*.C  C:\MSC
```

运行“d4learn”

程序“d4learn.exe”存在磁盘目录“\EXAMPLES”下，用于帮助学习 Code Base。使用下拉菜单选择 Code Base 例程。当“d4learn”被标识为例程的参数，这个例程开始执行并把结果显示出来。

在磁盘目录“\EXAMPLES”下，另外还有一些作为例子的程序。请参考文件“\EXAMPLES\EXAMPLES.DOC”。

0.6 生成 Code Base 应用程序

在 Code Base 应用程序源文件中，经常要用到下述“#include”伪指令：

```
#include (d4base.h)
#include (w4.h)
```

如果 Code Base 头文件不是放在 include 目录下，而是在 Source 目录下，则使用下述“#include”伪指令：

```
#include "d4base.h"
#include "w4.h"
```

文件“d4base.h”为 Code Base 数据管理例程定义结构和模式，“w4.h”为 Code Base 屏幕管理例程定义结构和模式。

例：

```
/* A similar example is in file "d4example.c" */
#include "d4base.h"
#include "w4.h"

#ifdef TURBO
    extern unsigned _stklen = 10000;
#endif

main()
{
    int j; long ref;

    /* Initialize Code Base and clear the screen */
    d4init();
    w4clear(-1);

    /* Open the database */
    if ( d4use("d4learn") < 0 ) exit(1);

    w4(0,5,"D4LEARN.DBF Field Names:");
    for ( j=1; j<= f4num_fields(); j++)
    {
        ref = f4j_ref(j);

        /* Display the field name */
        w4(j,8, f4name(ref));
    }
```

```
    w4cursof( j,0 ) ;

    w4exit( 0 ) ;
}
```

0.7 常见的问题

问题主要分为四个方面，即编译、连接、追踪和理解。

编译

编译一个 Code Base 源程序或实例文件时，用户有时会遇到编译错误，通常是由于忘记设置适当的编译转移点而引起的。例如，Turbo C 用户经常忘记设置"TURBO"条件编译转移点。不同的编译程序往往使用不同的头文件，有不同的运行时间库，Code Base 用条件编译转移点来解决这些差别。

遇到复杂的编译应用程序代码，要尽量予以简化。把出错的代码行转换成测试文件。先删去测试程序中通不过编译的部分。这种办法往往能准确地把问题分离出来。

连接

大多数模糊的连接错误，象"undefined symbol__fcvt87"或"fixup overflow"都是由编译不一致引起的。例如，修改了存贮模型而没有对目标模块进行完整的再编译，或者，库文件中的目标模块也许没有建立与应用程序相同的存贮模型。另一种编译不一致的方法是对一些编译过程进行浮点仿真，而对另一些编译过程使用不同的浮点数。这种一致的编译规则对库文件中的目标模块也同样适用。即库文件中的目标模块与应用程序用同样的方式编译。

另一个引起外部代码未定义的原因是忘记连适当的库或目标模块，也许是调用了不存在的例程。

所有第二个字符是数字 4 的外部过程名有可能是 Code Base 定义的外部过程，所有 Code Base 外部过程名都遵循这一约定。Code Base 内部使用的外部变量大部分在文件"d4init.C"中定义。有时也会有一个未定义的 Code Base 变量，这是由于没有调用初始化 Code Base 的例程。例如，如果仅调用"d4seek"，变量'v4cur__base'就成为未定义的外部过程。如果在调用'd4seek'之前调用了'd4use'则'v4cur__base'就被定义了。这是因为'd4seek'引用了一个包含在文件'd4init.c'中的初始化例程，并且与'd4init.c'对应的目标模块变成应用程序的一部分。变量'v4cur__base'是在文件'd4init.c'中被说明的。

追踪

如果 Code Base 程序不象予期的那样工作，有许多方法可用于确定问题。

1. 让编译程序查错，便用完全原型的最大警告级编译。
2. 分离

   把应用程序中不能工作的部分分隔出来，并独立地运行这部分程序。
3. 先发现的问题先解决

   研究首先发现的问题，第二个问题有可能是由第一个问题引起的。
4. 确定出错点

如果一个正在运行的例程中止了运行，工作存贮区有可能被破坏，这时要确定程序在何处中止运行，这一点可能就是错误发生之处。

有些故障可能十分微妙。例如，例程 A 可能破坏例程 B，而例程 B 又破坏例程 C，而例程 C 被破坏是在它第五次被调用时发生。

如果在 Code Base 中发现了错误，可用 'd4learn. exe' 来尝试重现这个错误，这一信息是 Sequiter 软件公司为支持工作人员确定问题的原因而提供的帮助，它可以帮助工作人员确定问题是与数据有关，还是一个故障。'd4learn. exe' 是用户本人用于查错的一个出色工具。例如，当使用 'd4learn. exe' 时，如果 'd4go' 报告一个错误，表明用户企图读的内容不存在。

理解 Code Base

如果感到理解 Code Base 的某一部分有困难，首先应确认你所阅读的材料是否切题，同时审阅实例程序并用 'd4learn. exe' 验证。

每个模块开始有一个引言，所包含的信息适用于该模块的所有例程，这些引言都很短，建议一读，不要跳过。

0.8 分类例程表

这里是一个按分类排列的例程表，其中仅包含较常用的例程。

一、数据库管理类

扩充、写、插入

d4append, d4write, x4insert

字段存取

f4char, f4double, f4int, f4long, f4memo, f4ptr, f4str, f4record

字段替换

f4r-char, f4r-double, f4r-int, f4r-long, f4r-str

字段信息

f4decimals, f4j-ref, f4num-fields, f4name, f4record-width, f4ref, f4type, f4width

过滤

x4filter, x4filter-reset, x4filter-do, i4filter

文件的打开、关闭、建立

f4create, d4use, d4use-excl, i4open, i4index, u4open

锁定、解锁例程

d4lock, d4locked, d4lock-all, d4lock-code, d4lock-wait, d4unlock, i4unlock, u4lock, u4unlock

备注文件

m4delt, m4exist, m4read, m4write

压缩、扩展、再定位

d4pack, d4zap, i4reindex

定位

d4bottom, d4go, d4seek-double, d4seek-str, d4skip, d4top, x4bottom, x4go, x4seek, x4skip, x4top

关系

x4relate, x4relate-do, x4relate-reset

二、屏幕管理类

颜色/属性

g4attribute, n4attribute, n4attribute-item, w4attribute, w4num-att

数据入口

g4, g4double, g4picture, g4read, g4upper, g4valid, b4browse, b4edit

菜单调用的数据入口

g4call, n4get-calc, n4menu, n4menu-help

显示

u4error, w4, w4box, w4clear, w4display, w4double, w4field, w4int, w4long, w4num, x4list

定义菜单

n4, n4action, n4activate, n4calc, n4item, n4reaction

窗口的激活、关闭、定义

w4activate, w4deactivate, w4define, w4close

窗口特性

w4border, w4handle, w4height, w4memory, w4popup, w4title, w4width

第一章　例程说明

引　言

本章解释所有 Code Base 外部例程，这些例程通过用户程序调用。

为了有效地利用 Code Base 的数据库管理部分，最好熟悉所有的 Database 和 field 例程和一些 index file 例程。另外，extended 例程对希望使用关系和过滤功能的用户也是很重要的。

对屏幕管理功能，应参考 Windowing、menuing 和 get 例程。首先学习 windowing 例程的用法。Menuing 和 get 例程依赖于 windowing 例程给予的能力，例如下拉菜单和全屏幕数据入口。

'conversion'、'utidity'、'momory handling'和'expression evaluation'例程一般应用程序员都会用到。

那些希望快速、一般了解 Code Base 例程的读者，可阅读例程部分的简介然后用'd4learn'程序作验证。

1.1　Browse/Edit 例行程序

browse 例程使用 Code Base 的数据库和屏幕管理两部分来浏览和/或编辑屏幕以输入和察看数据。

由于命令界面是固定的，使用这些例程可以为用户提供一致的接口，然而每个屏幕格式是可由用户设计的。

在 Code Base 盘子目录'BROWSE"下，有一些无权终端用户文件，文件可以帮助软件开发者快速为用户生成文件。

edit 屏幕同一时刻可以编辑一个记录，browse 屏幕同时可浏览多个记录，browse/edit 可在二者之间快速转换。

brows/edit 屏幕对终端用户十分有用。例如终端用户可用 brows 屏幕快速考查多个记录然后通过敲一个按键，便 edit 窗口出现。这时终端用户可仔细研究一个记录信息。

使用 browse 例程前应学会使用 data entry 和 windowing 例程。

b4browse

用　法　int b4browse(int (* browse__define) (int),
int (* edit__define)(int))

说　明　此例程为当前被选数据库生成并活化一个浏览窗口。
请参考下面的例子。

参　数

名称	用法
browse__define	是指向一个例程的指针。该例程提供一个可由用户定义的 browse 屏幕格式。 请参考下面的例子。

edit__edfine　　指向一个例程的指针，该例程提供一个可由用户定义的 edit 屏幕格式。如果这个参数为 0 或空，则无 edit 屏幕。请参考例程 'b4edit'

返回值　'(int)－1'表示一个错误的结果，反之，返回值为'(int)0'。

锁　定　被选数据库和它的索引文件在等待任何来自用户的输入时，都处于解锁状态。返回之前也被解锁。

例　子

```
#include <d4base.h>
#include <w4.h>
static int browse__setup(void) ;
static int browse__setup()
{
  /* The first few lines of the browse setup routine
     defines an appropriate window. */
  w4define( 1,0,24,79 ) ;
  w4border( DOUBLE, F__WHITE ) ;
  w4title( 0,－1, " Customer Database rowsing ",
        F__WHITE ;
  /* These next three routine calls must be present in
     every browse setup routine. */
  w4memory() ;
  w4activate(－1) ;
  g4release(0);
  /* Now the screen layout is defined. Note that
     any ´ Get´ routine except ´ g4read´ can be called.
     Refer to the ´ Get Routines´ chapter.
  */
  /* When calling a ´ Get Initialize´ routine from a
     browse setup routine, make the first parameter
     ´ (int) －1´ . This is the ´ row´ parameter which
     is set by Code Base under browse screens. */
  w4( 1,2, "Name" ) ;
  g4field( －1,4, f4ref("COMPANY" ) ) ;

  /* The window reference number of the defined window
     is returned. */
  return( w4select(－1) ) ;
}
main()
```

```
{
  d4use( "CUSTOMER" )
  i4open( "NAME" ) ;

  b4browse( browse__setup, 0 );

  d4close__all () ;
  w4exit(0) ;
}
```

b4call

用　法　void b4call(int (* call__routine)(int,int)

说　明　此例程指定一个例程 'b4edit' 和 'b4browse',在显示一个记录或把一个修改过的记录写入磁盘之前调用该例程。这个调用例程的功能类似于显示计算数据。调用例程有两个整型参数和一个整型返回值。参数之一是 browse/edit 窗口的引用号,第二个参数是记录显示在屏幕上的行号。edit 窗口的第二个参数值总是为 0。调用例程的返回值必定为 0。

例　子

```
#include <d4base.h>
#include <w4.h>
long a__ref, b__ref ;
static int setup(void)
{
  int w__ref ;
  w__ref = b4quick__browse() ;
  /* Like all browse setup routines,   b4quick__browse'
     activates the browse window. Consequently, it
     is safe to call ' w4' to display an extra title.
     */
  w4( 1, 40, "Total" ) ;
  return w__ref ;
}

static int calc( int w__ref, int row )
{
  w4double(                                                    row,40,
  f4double(a__ref)+f4double(b__ref),10,2) ;
}
main()
```

```
    d4use( "DATA" ) ;
    a_ref = f4ref( "A_VALUE" ) ;
    b_ref = f4ref( "B_VALUE" ) ;
  /* Call 'b4call' anytime before calling 'b4browse'
     */
    b4call( calc ) ;
    b4browse( setup, 0 ) ;
    d4close_all() ;
    w4exit(0) ;
}
```

b4edit

用　法　int b4edit(int (* browse_define)(void),
　　int (* edit_define)(void))

说　明　此例程为当前被选数据库生成并活化一个编辑窗口，注意如果两个参数都不为0，'b4edit'几乎与'b4browse'相同。不同之处仅是'b4edit'启动一个编辑屏幕，而'b4browse'启动一个浏览屏幕。

下面的例子出现在'BROWSE'子目录下，可以作为一个样本。

参　数

名称	用法
browse_define	是 指向一个例程的指针，该例程提供一个可由用户定义的浏览屏幕格式。如果这个参数为0或空，则无浏览屏幕。 参考例程'b4browse'
edit_define	是 指向一个例程的指针，该例程提供一个可由用户定义的编辑屏幕格式。 参考下面的例子。

返回值　'(int) -1'表示一个错误的结果，反之，返回值为'(int)0'。

锁　定　被选数据库和它的索引文件在等待任何来自用户的输入时，都处于解锁状态，返回之前也被解锁。

例　子

```
#include <d4base.h>
#include <w4.h>
static int edit_setup(void) ;
static int edit_setup()
{
  /* The first few lines of the edit setup routine
     defines an appropriate window. */
  w4define( 1,0,24,79 ) ;
  w4border( SINGLE, F_WHITE ) ;
  w4title( 0,-1, " Editing Customer Database ",
```

```
            F__WHITE ;
    /* These next three routine calls must be present in
       every edit setup routine. */
    w4memory() ;
    w4activate(-1) ;
    g4release(0);
    /* Now the screen layout is defined. Note that
       any 'Get' routine except 'g4read' can be called.
       Refer to the 'Get Routines' chapter.
    */
    w4( 1,2, "Name" ) ;
    g4field( -1,12, f4ref("NAME" ) ) ;
    w4( 3,2 "Company") ;
    g4field( 3,12, f4ref("COMPANY") ) ;

    return( w4select(-1) ) ;
}
main()
{
    d4use( "CUSTOMER" )
    i4open( "NAME" ) ;

    b4edit( 0, edit__setup ) ;

    d4close__all () ;
    w4exit(0) ;
}
```

b4margin

用　法　void b4margin(int top__margin, int bottom__margin)

说　明　此例程为浏览例程'b4browse'设置上下边缘。例如，从美观考虑，浏览窗口的顶部和低部都应留有一定的空格。'b4browse'的缺省值是顶部三行和底部一行不显示。通过调用'b4margin'可以改变缺省值。

参　数

名称	用法
top__margin	窗口顶部空行数，'b4browse'不在这几行内显示。
bottom__margin	窗口底部空行数，'b4browse'不在这几行内显示。

例　子　b4margin(2,4) ;

b4quick__browse

用　法　int b4quick__browse(void)

说　明　此例程是一个编辑功能设置例程，可应用于任何数据库。此例程从不直接调用，而是通过'b4edit'或'b4browse'的第一个参数传递。

例　子　b4browse(b4quick__browse, 0) ;

b4quick__edit

用　法　int b4quick__edit(void)

说　明　此例程是一个编辑功能设置例程，可应用于任何数据库。此例程从不直接调用，而是通过'b4edit'或'b4edit'的第二个参数传递。

例　子　b4edit(0, b4quick__edit) ;

b4verify

用　法　void b4verify(int (* verify__routine) (int))

说　明　' b4call' 指定的一个例程，该例程被'b4edit'和'b4brows'调用，用于输入数据检验。

如果此"数据检验"程序返回值不为0，表明继续输入数据，记录未写。数据检验例程的参数是当前编辑/浏览的窗口引用号。

例　子

```
#include <d4base.h>
#include <w4.h>
long a__ref,b__ref ;
static int check() ;
static int check( int w4__ref )
{
  if ( f4double(a__ref) <0.0 || f4double(b__ref) < 0.0
  )
  }
    w4display( "Entry Error",
               "All values must be positive", (char
               *) 0 ) ;
    return -1 ;
  }
  return 0 ;
}
main()
{
  d4use( "DATA" ) ;
```