C语言与数据库管理系统接口软件

CODE BASE

使用指南

廖建勇 编著

国防科技大学出版社

C语言与数据库管理系统接口软件

CodeBase使用指南

○廖建勇　编著

国防科技大学出版社

图书在版编目(CIP)数据

CodeBase 使用指南/廖建勇　编著—长沙:国防科技大学出版社,1997.5
ISBN 7—81024—415—9

Ⅰ　CodeBase 使用指南
Ⅱ　廖建勇
Ⅲ　CodeBase-数据库-C 语言
Ⅳ　TP311.13

责任编辑：卢天贶
责任校对：谷建湘
封面设计：陆荣斌

[redacted]国防科技大学出版社出版发行
[redacted]5[redacted]681　邮政编码：410073
新华书店总店北京发行所经销
长沙交通学院印刷厂印装
开本：787×1092　1/16　印张：15.25　字数：352 千
1997 年 5 月第 1 版第 1 次印刷　印数：1—2000 册

ISBN 7—81024—415—9
TP.85　定价：18.00 元

内 容 简 介

CodeBase 是 C 语言、dBASE 及与之兼容的 FoxBase⁺、FoxPro 系统的接口软件，使用 CodeBase 可以十分方便地实现 C 语言与数据库管理系统的混合编程。

本书首先介绍了 CodeBase 的基本概念和有关数据库的基本知识，然后详细地介绍了 CodeBase 各个函数的意义与用法，并给出了函数使用范例。最后给出了利用 CodeBase 系统实现 C 语言与 FoxPro 系统混合编程的应用实例。

本书可作为软件开发工作者的参考书，也可作为大专院校教学参考书。

前　言

随着微型计算机的迅速发展与普及，计算机的应用已从科学计算、实时控制等传统领域迅速扩展到非数值处理的各个领域。

数据库管理系统是帮助人们处理大量信息，实现管理科学化和现代化的强有力工具。由于数据库具有数据结构化、最低冗余度、易于扩充、易于编制应用程序等特点，因而成为近年来发展最快的计算机软件之一。在微机上实用的关系型数据库系统有 dBASE、FoxBase⁺、FoxPro、Date、Knowledge、E_MAN、BASEⅣ、RBASE 等等。其中以 dBASE 及与之兼容的 FoxBase⁺、FoxPro 系统应用最为广泛，是最具有发展前途的一种数据库管理系统。

C 语言是美国贝尔实验室于 1972 年推出的一种高级计算机语言。由于 C 语言具有使用灵活方便、数据类型及运算符丰富、可移植性好、执行效率高、能直接访问计算机的硬件资源、具备高级语言和汇编语言的优点、既适合于编写应用软件又适合于编写系统软件等特点，经过短短二十几年的发展，已成为当今世界使用最为广泛的计算机高级语言。

综合运用 C 语言强大的运算功能以及执行效率高的特点和 dBASE、FoxBase⁺、FoxPro 系统强大的数据处理能力已成为应用软件开发的一种手段。在邮电、通讯、证券交易所等要求执行速度快、实时性强、计算量大的应用领域中，使用 C 语言和 dBASE、FoxBase⁺、FoxPro 系统混合编程是软件开发的必然趋势。美国 Sequiter 软件开发公司于 80 年代推出了 C 语言与数据库系统的接口 CodeBase 系统。CodeBase 是管理数据库的 C 语言函数库，它具有以下特点：

（1）与 dBASE、FoxBase⁺、FoxPro 等数据库管理系统兼容。

利用 CodeBase 系统，程序员可以直接用 C 语言建立、读取或修改 dBASE、FoxBase⁺、FoxPro 系统的数据库文件、索引文件和记忆文件。

（2）具有多用户功能，可以在网络上执行。

CodeBase 具有完整的多用户功能。当应用程序在网络上运行时，CodeBase 会自动检测并利用网络资源。

（3）可以直接被 C 语言所调用。

CodeBase 系统实质上是一组可以直接调用和处理数据库文件的 C 语言函数库，因此 C 程序员可以在一个相当高的水平上使用所有 CodeBase 的函数。利用 CodeBase 所编写的应用软件执行速度快、可读性好、且没有不必要的代码。

（4）占用内存少。

一般说来，CodeBase 系统所占用的内存不会高于 1K 字节，而且可以通过计算直接求出。

（5）对运行环境的要求低。

理论上讲，只要用户具有C语言的编译器就可以使用CodeBase系统。

(6) 易学、易用、易掌握。

只要用户具有一定的C语言编程知识，就可以通过实际例题很快学会并掌握CodeBase系统。

在我国经济较为发达的东部沿海地区，已有大量软件工作者使用CodeBase系统。但是到目前为止，还没有一本详细介绍CodeBase软件的专著。编写本书旨在填补这一空白，促进这一软件在我国的推广与使用。

本书得以出版，离不开株洲工学院院领导和计算机系领导的大力支持，作者在此深深地感谢各位领导的关心与支持。在本书的编写过程中，始终得到我的妻子刘艳女士的支持与帮助，本书的录入和排版工作主要是由她来完成的，在此，作者向她致以崇高的敬意。

书中所有例题都在Borland C++及486型微机上调试通过，并有配套软盘，需要者请直接与作者联系。联系电话：0733--8100057（宅），0733－8100200（办），联系地址：(412008) 湖南省株洲工学院计算机系。

作　者

1996年于株洲工学院

目　录

第一章 概 述

§1.1 CodeBase 的特点

随着微型计算机的迅速发展与普及，计算机的应用已从科学计算、实时控制等传统领域迅速扩展到非数值处理的各个领域，如计算机辅助设计、通讯、人工智能等。应用软件的开发也从单一语言编程发展到多语言混合编程。

数据库管理系统是帮助人们处理大量信息，实现管理科学化和现代化的强有力工具。由于数据库具有数据结构化、最低冗余度、较高的程序独立性、易于扩充、易于编制应用程序等特点，因而成为近年来发展最快的计算机软件之一。目前，数据库技术已广泛应用于文化教育、科学计算、人工智能、计算机辅助设计与制造、通讯、邮电、银行及证券行业中。

在微机上实用的关系型数据库系统有 dBASE、FoxBase⁺、FoxPro、Date、Knowledge、E_MAN、BASEIV、RBASE 等等。其中以 dBASE 及与之兼容的 FoxBase⁺、FoxPro 系统应用最为广泛，是最具有发展前途的一种数据库管理系统。

dBASE、FoxBase⁺、FoxPro 系统具有功能强大、操作方便、易于学习与使用等特点，可以在 DOS、Windows 等操作系统中运行。同时，由于 dBASE、FoxBase⁺、FoxPro 系统提供了在网络环境下运行所需要的附加命令，提供了共享文件的加锁和共享记录的加锁两级保护，从而实现了数据共享和安全保密性。

C 语言是美国贝尔实验室于 1972 年推出的一种高级计算机语言。由于 C 语言具有使用灵活方便、数据类型丰富及运算符、可移植性好、执行效率高，能直接访问计算机的硬件资源，同时具备高级语言和汇编语言的优点，既适合于编写应用软件又适合于编写系统软件等特点。因此，经过短短二十几年的发展，C 语言已成为当今世界使用最为广泛的计算机高级语言之一。目前最为流行的关系型数据库 dBASE、FoxBase⁺、FoxPro 系统以及 UNIX 操作系统就是由 C 语言编写的。

综合运用 C 语言功能强大以及执行效率高的特点和 dBASE、FoxBase⁺、FoxPro 系统强大的数据处理能力已成为开发应用软件的一种手段。在邮电、通讯、证券交易所等要求执行速度快、实时性强、计算量大的应用领域中，使用 C 语言和 dBASE、FoxBase⁺、FoxPro 系统混合编程是软件开发的必然趋势。

为了在 C 语言环境中调用数据库，实现 C 语言与 dBASE、FoxBase⁺、FoxPro 系统的混合编程，用户必须了解并掌握数据库的存储结构，并据此对其进行操作。这样的混合编程势必造成重复劳动，降低程序的执行效率，浪费计算机的资源，大大降低软件的开发

速度。

为克服以上缺陷，美国 Sequiter 软件开发公司于 80 年代推出了 C 语言与数据库系统的接口 CodeBase 系统。CodeBase 是管理数据库的 C 语言库，它具有以下特点：

（1）与 dBASE、FoxBase⁺、FoxPro 等数据库管理系统兼容

利用 CodeBase 系统，程序员可以直接用 C 语言建立、读取或修改 dBASE、FoxBase⁺、FoxPro 系统的数据库文件、索引文件和记忆文件，可以很方便地对数据库进行筛选（SELECT）、投影（PROJECT）、联接（JOIN）等各种操作。

特别地，CodeBase 系统将记忆型字段当成字符型字段进行处理，从而大大简化了记忆类型字段的处理过程，克服了 dBASE、FoxBase⁺系统处理记忆类型字段的不足。

（2）具有多用户功能，可以在网络上执行

CodeBase 具有完整的多用户功能。当应用程序在网络上运行时，CodeBase 会自动检测并利用网络资源。CodeBase 有两种对记录或文件加锁的方式：外部加锁（由应用程序完成）和自动加锁（由系统自动完成）。而且，CodeBase 系统可以忽略对文件或记录所加的锁而直接读取已被另外的用户加锁的数据。

（3）可以直接被 C 语言所调用

CodeBase 系统实质上是一组可以直接调用和处理数据库文件的 C 语言函数库，因此 C 语言程序员可以在一个相当高的水平上使用所有 CodeBase 的函数。利用 CodeBase 所编写的应用软件执行速度快、可读性好，且没有多余的代码。

（4）占用内存少

一般说来，CodeBase 系统所占用的内存不会高于 1K 字节，而且可以通过计算直接求出（请参考本书的第七章）。

（5）对运行环境的要求低

从理论上讲，只要用户具有 C 语言的编译器就可以使用 CodeBase 系统。但是作者建议用户在 486 机型、4M 以上硬件环境中使用 CodeBase 系统。

（6）易学、易用、易掌握

只要用户具有一定的 C 语言编程知识，就可以通过实例很快学会并掌握 CodeBase 系统。具有一定的数据库系统知识对使用 CodeBase 是非常有益的但不是必要的。

§1.2 CodeBase 的安装

当你需要在 DOS、Microsoft Windows 以及 OS/2 等操作系统中安装 CodeBase 时，请按下面的步骤进行安装，安装完毕后请查看磁盘上的“Sread. me”文件。

（1）将当前驱动器改为你期望安装 CodeBase 的驱动器（如 C:，D:等）；

（2）将 CodeBase 的 1# 系统盘插入软驱，并执行其上的 DOS 系统可执行文件“install. exe”；

（3）也可以从光盘上直接安装 CodeBase 系统。

例如：

D>C:

C>B: INSTALL

这是将 CodeBase 从软驱 B 装入硬盘 C.

安装过程中，你首先需要选择 CodeBase 系统的路径；然后就可以有选择地安装 CodeBase 系统。你也可随时安装 CodeBase 的一部分。

CodeBase 的源代码、头文件以及特殊编译器文件（如库文件）将会安装到缺省目录中，不过，在安装过程中可以修改缺省安装路径。

为了使用 CodeBase，通常只需要从安装盘上装入适当的头文件和库文件。在某些情况下，用户可能需要建立一个自己的库文件，此时，就必须安装 CodeBase 的源代码。

你如果选择安装一种特殊的编译系统，那么与该编译系统有关的文件就会安装到缺省目录中。可以检查缺省目录中的“Sread. me”文件以获得这些文件的名称。

你的编译器如果不直接支持 CodeBase，而是一个支持 ANSIC 的编译器，则你也可以使用 CodeBase. 然而使用的方法却有所不同，此时你需要做以下工作：

（1）安装头文件和源代码；

（2）将 CodeBase 的源代码编译成目标文件；

（3）将目标文件编译成库文件；

（4）将库文件加到编译器的库路径中。

CodeBase 系统为每一个它所支持的编译器都提供一个供编译的命令文件和一个库文件，而且该库文件可以由用户重新编译生成。为了将 CodeBase 用于它所不支持的 C 语言编译器上，可以修改该命令文件，然后再在该编译器中对 CodeBase 的源代码进行编译，生成该编译器中可以使用的库文件。在购买时应检查 CodeBase 是否支持你所希望的操作系统及 C 语言编译器，以免引起不必要的麻烦。

§1.3 CodeBase 应用程序简介

在 CodeBase 系统盘上可以安装许多应用实例。下面举其中的两个例子。

例一：显示一个数据库的内容

此应用实例打开一个数据库文件，将记录指针定位到每一个记录，并进行显示。#include<d4all. h>

所有的 CodeBase 函数原型都定义在所包含的头文件中。如果需要更多的信息，请参考本书第五章。

```
extern unsigned_stklen=100000;    //Borland only
```

这一行是为 Borland 公司的 C 语言编译器定义堆栈大小。对其它编译器，在连接应用程序时会自动定义堆栈大小。如果 CodeBase 系统或应用程序所使用的堆栈超过了该预定值，而在生成 CodeBase 库文件时堆栈检测标志没有置成真（非零值），则可能产生不可预见的结果。为了更多地了解堆栈空间，请参考有关专著。

```
C4CODE  code_base;
```

此行定义一个“C4CODE”结构体变量，并为它分配内存单元。通常，一个应用程序具有唯一一个“C4CODE”结构体变量。该结构体包含 CodeBase 的各种设置以及应用程序

中需要使用的、指向内存单元的指针。下面的“d4init(&code_base);”行用于初始化该CodeBase结构体中的各种设置。在调用其它的CodeBase函数之前应调用“d4init ()”函数。请参考第五章中有关C4CODE变量的说明。

```
D4DATA * base;
```

该行定义一个“D4DATA”结构体指针。调用“d4open ()”函数所返回的指针就保存在该结构体指针中。在应用程序中，对每一个数据库文件都应设置一个相应的D4DATA结构体指针，在使用数据库的数据之前，应该用“d4open ()”函数将该数据库文件打开。

```
d4init (&code_base);
base=d4open (&code_base, "rsgl");
e4exit_test (&code_base);
```

调用函数“e4exit_test ()”的目的是检查“d4open ()”函数是否真正打开了一个数据库。如果没有，“e4exit_test ()”将中止程序运行，返回操作系统，并给出错误提示，如果运行本例时当前目录中没有一个“RSGL.DBF”的数据库，则可能出现如下错误提示：

```
Error Number  -60
Opening File
File Name
RSGL.DBF
Press a key...
```

```
for (d4top (base);! d4eof (base); d4skip (base, 1))
```

这是外循环。它从数据库文件的第一个记录开始对数据库的每一个记录进行检索。每结束一次循环，就将数据库中一个不同的记录读入到记录缓冲区中。注意，如果出现了错误，则“d4eof ()”会产生一个错误代码，它使得条件“! d4eof (base)”的值变为假，从而中止循环。

```
for (j=1; j<=d4num_fields (base); j++)
```

这是内循环。内循环检测数据库中某一条记录的每一个字段，并显示每一个字段的内容。内、外循环结合就可以在屏幕显示数据库文件的每一个记录的每一个字段的内容。

```
field=d4field_j (base, j);
printf ( "%s", m4str (field)); //Display the field
```

首先通过调用函数“d4field_j ()”为一个F4FIELD结构体指针赋值，使之指向一个特定的字段，然后显示该字段的值。这个F4FIELD指针是打开数据库时由CodeBase系统自动进行分配的。因为数据库中有可能出现记忆类型字段，所以使用了“m4str ()”函数而不是“f4str ()”函数。另外，由“m4str ()”函数所返回的值是一个带字符串结束标志的指针，这正是调用“printf ()”函数所需要的。“printf ()”函数是C语言编译器提供的一个库函数，一般的C语言编译系统都会提供该函数。

```
d4close_all (&code_base);
```

“d4close_all ()”关闭所有已打开的数据库文件并释放它们所占用的内存单元。因为此例中只打开了一个数据库文件，所以，可以调用“d4close ()”来完成这一工作。

```
e4exit (&code_base);
```

这最后一行的作用是返回到操作系统中。如果产生了错误，则将该错误传递到操作系统，并在屏幕上显示。这一行并不是必须的，也就是说，可以不调用“e4exit（ ”函数。

下面是源程序清单：

```
#include<d4all.h>
extern unsigned_stklen=1000; //Borland only
int main (void)
{C4CODE code_base;
D4DATA * base;
F4FIELD * field;
int j;
d4init (&code_base);
/* open the data file. If there is a production MDX and/or memo
files, they are also automatically opened. */
base=d4open (&code_base, "rsgl");
e4exit_test (&code_base);
//Loop through the entire data file
for (d4top (base);! d4eof (base); d4skip (base, 1))
{  printf ( "\n");    //Display the record on a new line
//Loop through every field
for (j=1; j<=d4num_fields (base); j++)
{
field=d4field_j (base, j);
printf ( "%s  ", m4str (field));    //Display the field
}
}
d4close_all (&code_base);
e4exit (&code_base);
return 0;
}
```

运行结果分析：

假设磁盘上有一名为RSGL、DBF的数据库，其结构为：

序号	字段名	字段类型	字段长度	小数位数	意义
1	NAME	字符型(C)	10	0	姓名
2	BIRTHDAY	日期型(D)	8	0	出生日期
3	WAGE	数值型(N)	7	2	基本工资
4	MARRIED	逻辑型(L)	1	0	婚否

数据库的内容如下：

记录号	姓名	出生日期	基本工资	婚否
1	张爱明	12/03/64	286.30	.T.

```
2       刘虹辉   05/12/23   874.40     .T.
3       李  强   10/12/76   240.00     .F.
```

执行以上程序将在屏幕上显示以下信息：

```
张爱明   12/03/64   286.30   T
刘虹辉   05/12/23   874.40   T
李  强   10/12/76   240.00   F
```

例二：从 ASCII 文件中追加记录

此例建立一个数据库文件，并且从一个 ASCII 码文件中读入数据，形成数据库的一个记录。

```
#define DESC_LEN        20
#define CODE_LEN        4
#define VALUE_LEN       6
#define QUANTITY_LEN    5
```

ASCII 码文件的结构基本上由以上四条" define" 语句来说明。首先有一个长度为 20 的货物名称，然后是一个长度为 4 的货物代码，接着是一个长度为 6 的货物价值，最后是一个长度为 5 的货物数量。在每一个输入行的末尾是一个任意的字母后接一个行结束符（换行符）。注意，以上所述的长度都是指英文字符数，如果其中出现汉字，则每个汉字占用两个字符宽度。

```
#define DESC_POS        0
#define CODE_POS        (DESC_POS+DESC_LEN)
#define VALUE_POS       (CODE_POS+CODE_LEN)
#define QUANTITY_POS    (VALUE_POS+VALUE_LEN)
```

接下来的四条“define”语句用于计算每一个数据项的位置。例如，有关“CODE”的数据是从一行 ASCII 文件的第 20 列开始的。

本例中的大部分变量都被定义为全局变量而非局部变量。这样做的目的是向你介绍另外一种定义变量的方法。事实上，完全可以用局部变量来完成本例的所有工作。

```
inventory=d4create (&code_base," INVENTOR", inventory_fields, 0);
```

变量“inventory_fields”用于定义将要建立的数据库文件的结构。注意：数据库文件中的记录型式与 ASCII 文件中的数据是不相同的。这样做的目的是为了增加例题的趣味性。

数据库文件中的“DATE”字段与 ASCII 码文件没有什么联系，事实上，“DATE”字段是由计算机系统当前日期决定的。

```
h4open (&ascii, &code_base, "inventor. asc", 0);
4exit_test (&code_base);
```

调用函数“h4open ()”打开提供数据的 ASCII 码文件，接着调用函数“e4exit_test ()”以检测数据库文件是否已正确建立以及 ASCII 文件是否已真正打开。

descrip=d4field (inventory, "DESCRIP");

code =d4field (inventory, "CODE");

date =d4field (inventory, "DATE");

value =d4field (inventory, "VALUE");

quantity=d4field (inventory, "QUANTITY");

这几个语句重复调用函数"d4field ()"，对数据库文件的字段指针初始化。这些字段指针（date 除外）将根据 ASCII 文件的内容进行赋值。

h4seq_read_init (&seq_read, &ascii, 0, seq_read_buf, sizeof (seq_read_buf));

调用函数"h4seq_read_init ()"，对 H4SEQ_READ 类型结构体变量"seq_read"进行初始化。调用此函数的目的是使应用程序可以采用读数据块的方式从 ASCII 文件中读取数据，因为这样做可以将读数据的速度提高 30 倍，从而大提高程序的执行效率。注意，也可以用块写方式来产生数据库文件（后缀为.DBF），然而，程序编写却要复杂得多。

while(1)

"while ()"的每一次循环都从 ASCII 文件中读取一组数据，并将它作为一条记录添加到数据库文件中。当 ASCII 文件中没有另外的数据或者产生了错误时，通过条件

if (len_read! =sizeof (ascii_buf)) break;

从循环中退出。

d4append_start (inventory, 0);

f4assign_n (descrip, ascii_buf+DESC_POS, DESC_LEN);

f4assign_n (code , ascii_buf+CODE_POS, CODE_LEN);

f4assign_n (date , today , 8);

f4assign_double (value, c4atod (ascii_buf+VALUE_POS, VALUE_LEN));

f4assign_double (quantity, c4atod (ascii_buf+QUANTITY_POS, QUANTITY_LEN));

调用函数"d4append_start ()"是通知 CodeBase 系统，在这以后对数据库记录指针的修改是用于追加记录而不是修改当前记录，因此，系统就不会自动存盘。然后调用"f4assign_n ()"和"f4assign_double ()"将记录缓冲区的各字段用 ASCII 文件中相应的值来替代。在替代数值型字段时，使用的是"f4assign_double ()"函数，用此函数可以检测 ASCII 文件中数值是否合法。

d4append (inventory);

当把 ASCII 文件的一行中所有的数据都复制到记录缓冲区以后，就调用"d4append ()"函数，在数据库中追加一条记录。在完成记录的添加工作后，循环体结束，接着读取 ASCII 文件中下一行的数据，直到读完所有的数据为止。

d4close (inventory);

h4close (&ascii);

最后，在循环执行完毕后，应该将数据库文件和 ASCII 文件都关闭。调用"d4close ()"函数与调用"d4create ()"函数相关，调用"h4close ()"函数则与调用"h4open ()"函数相关。

以下是源程序清单：

```
#include <d4all.h>
extern unsigned_stklen=10000;    //Borland only
#define DESC_LEN              20
#define CODE_LEN              4
#define VALUE_LEN             6
#define QUANTITY_LEN          5
#define DESC_POS              0
#define CODE_POS              (DESC_POS+DESC_LEN)
#define VALUE_POS             (CODE_POS+CODE_LEN)
#define QUANTITY_POS          (VALUE_POS+VALUE_LEN)
//The carriage return and line feed characters are includeed
#define TOTAL_LEN (QUANTITY_POS+QUANTITY_LEN+2)
C4CODE code_base;
D4DATA *inventory;
F4FIELD *descrip, *code, *value, *quantity, *date;
H4FILE  ascii;
H4SEQ_READ seq_read;
char  ascii_buf [TOTAL_LEN], seq_read_buf [1024];
F4FIELD_INFO inventory_fields [] =
{
{ "CODE", 'c', 4, 0},
{ "DESCRIP", 'c', 30, 0},
{ "DATE", 'd', 8, 0},
{ "VALUE", 'n', 8, 2},
{ "QUANTITY", 'n', 6, 0},
};
void main (void)
{
char *today;
d4init (&code_base);
//Create the data file; open the ASCII file.
inventory=d4create (&code_base, "INVENTOR", inventory_fields, 0);
h4open (&ascii, &code_base, "inventor.asc", 0);
e4exit_test (&code_base);
descrip=d4field (inventory, "DESCRIP");
code   =d4field (inventory, "CODE");
date   =d4field (inventory, "DATE");
value  =d4field (inventory, "VALUE");
quantity=d4field (inventory, "QUANTITY");
//Initialize to read the ASCII file with extra fast performance
h4seq_read_init (&seq_read, &ascii, 0, seq_read_buf, sizeof (seq_read_buf));
a4today (today);
```

```
while (1)
{
unsigned len_read;
//read one ASCII record
len_read=h4seq_read (&seq_read, ascii_buf, sizeof (ascii_buf));
if (len_read! =sizeof (ascii_buf))
break;
//Start the Append Operation
d4append_start (inventory, 0);
//assign the fields
f4assign_n (descrip, ascii_buf+DESC_POS, DESC_LEN);
f4assign_n (code  , ascii_buf+CODE_POS, CODE_LEN);
f4assign_n (date  , today  , 8);
f4assign_double (value, c4atod (ascii_buf+VALUE_POS, VALUE_LEN));
f4assign_double (quantity, c4atod (ascii_buf+QUANTITY_POS, QUANTITY_LEN));
//Append the new record
d4append (inventory);
}
d4close (inventory);
h4close (&ascii);
e4exit (&code_base);
}
```

注意，运行此程序时，应该保证在当前目录中有一个名为“INVENTOR. ASC”的ASCII码文件，否则将出现以下错误提示：

```
Error Number  -60
Opening File
File Name
INVENTOR. ASC
Press a key...
```

同时应该保证有一个没有名为“INVENTOR. DBF”的数据库文件，否则将出现以下错误提示：

```
Error Number  -20
Creating File
File Name
INVENTOR. DBF
Press a key...
```

因为执行该程序后会产生一个名为“inventor. dbf”的数据库文件，因此在执行了一次该程序后，应该先删除此数据库文件，才能再一次执行该程序。为了避免这个问题，可以在程序中增加一个判断：当数据库文件存在时，就打开该数据库文件，并先删除其中所有的记录；如果数据库文件不存在，则先建立一个数据库（即程序中的方式）。具体作