用 C 语言直接读写数据库

刘奎亮　（保定变压器厂计算中心　071056）

摘要：本文介绍了如何用 C 语言直接读写数据库，以及需要注意的问题。

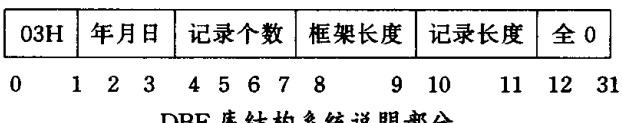
关键词：数据库　记录　结构框架

数据库的应用越来越广泛，以至于现在许多高级语言开发工具都增加了数据库引擎，也有些软件公司开发了供高级语言直接调用的数据库操作函数库。但是要完全掌握这些软件的使用并不是一件很容易的事，而且这些软件的使用也有一定的条件限制，比如常用的 C 语言数据库函数库 CodeBase，在 DOS 环境下可以正常运行，但在 AutoCAD 绘图系统中，欲与二次开发工具 ADS 冲突，Visual Basic 尽管也可以操作数据库，但一般限于 WINDOWS 环境。在 DOS 环境下，为了更有效、更方便地用高级语言直接读写数据库，必须先了解数据库结构，了解数据库记录是怎样存储的，另外还要注意用什么方式打开数据库，如何用高级语言自身的标准文件 I/O 函数读写数据库。用高级语言自身的 I/O 函数直接读写数据库一方面占用代码少、速度快，另一方面兼容性好。以下简要介绍一下用 Borland C 语言如何正确读写数据库。

首先要了解数据库的存储结构。

现在被广泛使用的数据库管理系统 dBASEIII、FOXBASE、FOXPRO 等都使用了相同的数据库结构，数据库文件（*.dbf）都是一种自说明的数据文件，由库结构说明框架和数据记录两部分组成，其中库结构说明部分又分为系统说明部分（占 32 个字节）以及字段说明部分（每个字段说明部分占 32 个字节）。

1. 数据库文件结构框架



字段名	0	类型		宽 度	小 数 位	全 0
0	9 10	11	12 13 14 15	16	17 18	31

DBF 字段说明部分

dBASE 数据库文件(*.DBF)的框架长度是不固定的，它随着字段定义个数的增加而增加，它的计算公式是：

库结构框架长度字节数 = DBF 库结构系统说明部分字节数(32) + 32 * 总的字段个数 + 1，框架部分的最后一个字节是框架结束符 0DH。

比如数据库有 5 个字段，则库结构框架长度为：32 + 32 * 5 + 1 = 193。

2. 数据记录

数据部分紧接着框架部分存放，字符型数据和数值型数据的空位均以空格形式存放，每个数据记录的第一个字节均为删除标志，若记录未被删除，则记录删除标志为空，即 0X20H，若删除则为字符'*'，如果数据库不为空，则整个文件的末尾有结束标记符 1AH。

一般情况下，在高级语言中需要随机读写数据库，此时数据库文件都是已存在的，所以必须以同时可进行读和写的方式打开数据库。

Borland C 系列打开标准 I/O 文件的方式：

"r"	打开文件进行只读操作，如果不存在则出错
"w"	打开文件进行只写操作，如果文件已经存在，则它的内容被破坏，否则 建立该文件。
"a"	打开文件在文件尾部进行写操作(就是添加)，如果该文件不存在，则建 立之。
"r+"	打开文件进行读和写操作，如果该文件不存在则出错。
"w+"	打开文件进行读和写操作，如果该文件已经存在，则它的内容被破坏，否则建立该文件。
"a+"	文件可以在任何位置读，但是只能在文件尾部写，如果该文件不存在，则建立之。

由上可见，若要在数据库文件的任意位置同时进行读和写操作，又不破坏原文件，则只能用"r+"方式打开数据库文件。另外由于数据库文件都是用二进制形式存储的，因此打开数据库文件时应该用"r+b"。即：

```
fopen((char *)file-name,"r+b");
```

比如打开 C:\FOX 目录下的数据库文件 myfile.dbf，则应用：

```
fopen("C:\FOX\myfile.dbf","r+b");
```

如果要打开的数据库在当前目录，则可省略路径名。

Borland C 语言提供了一套读写标准二进制文件的函数以及定位文件指针的函数，它们是：

```
fread(void *ptr, size-t size, size-t n, FILE *stream);
fwrite(void *ptr, size-t size, size-t n, FILE *stream);
fseek(FILE *stream, long offset, int whence); // 移动文件指针
```

其中：参数 whence 为 SEEK-SET，表示从数据库文件的开始处偏移

SEEK-CUR，表示从当前位置开始偏移 SEEK-END，表示从数据库文件的尾部向前偏移，关闭数据库用：

```
fclose(fp); // fp 为文件打开时返回的指针
```

读写数据库记录时，最好定义一个结构类型的变量，其长度要与数据库记录长度相同，并且按每个字段的长度在结构变量中定义相应的字符型数组，且顺序要与字段在记录中的次序一致。这里要注意一点：在许多 C 编译器中，结构通常按字(word)边界对齐。也就是说，用 sizeof() 函数返回的结构长度总为偶数。如果数据库记录长度为奇数，则定义结构变量时，不要把记录删除标志(1 个字节) 定义在内，以保证结构变量的长度为真正的偶数，这样每次读写数据库时，先读写记录删除标志，然后再用结构变量一次读写整个记录数据。如果数据库记录长度为偶数，则应把记录删除标志定义在结构变量中。

设数据库(test.dbf)结构如下：

Field	Field Name	Type	Width	Dec	Index
1	NAME	Character	6		
2	AGE	Numeric	2		
* * * Total * * *			9		

注意：数值型字段在数据库中是以字符形式存放的。

下列程序用 Borlandc 3.1 编译通过。

(假设数据库文件 test.dbf 已经在当前目录，且没有记录)

```
#include<stdio.h>
#include<string.h>
long rec-total=0;              // 记录总数
char file-end=0x1A;            // 文件结束标志
char rec-del-flag=0x20;        // 记录删除标志
struct {
  char name[6];
  char age[2];
} record;                      // 记录结构
void main()
{
FILE *fp;
int n, len;
fp=fopen("test.dbf", "r+b");  // 以随机读写方式打开数据库
fseek(fp, 8L, SEEK-SET);
fread(&len, 2, 1, fp);         // 得到数据库框架长度
fseek(fp, len, SEEK-SET);      // 移动文件指针到数据库记录的开始处
strcpy(record.name, "ABC");
len=strlen(record.name);
for  (n=5; n>=len; n--)  record.name[n]=0x20;   // 字段尾赋空字符
strcpy(record.age, "9");
len=strlen(record.age);
for  (n=1; n>=len; n--)  record.age[n]=0x20;    // 字段尾赋空字符
fwrite(&rec-del-flag, 1, 1, fp);                // 写删除标志
fwrite(&record, sizeof(record), 1, fp);         // 写记录
strcpy(record.name, "OK");
len=strlen(record.name);
for  (n=5; n>=len; n--)  record.name[n]=0x20;   // 字段尾赋空字符
strcpy(record.age, "25");
len=strlen(record.age);
for  (n=1; n>=len; n--)  record.age[n]=0x20;    // 字段尾赋空字符
fwrite(&rec-del-flag, 1, 1, fp);                // 写删除标志
fwrite(&record, sizeof(record), 1, fp);         // 写记录
fwrite(&file-end, 1, 1, fp);                    // 写文件结束标志 0x1A
fseek(fp, 4L, SEEK-SET);                        // 移动文件指针到记录个数处
rec-total=2;
fwrite(&rec-total, 4, 1, fp);                   // 写记录总数到结构框架
fclose(fp);                                     // 关闭数据库文件
}
```