

## 一. 文件基础操作

关于文件的几点说明：

1. 卡信息保存在结构体数组中，是在内存中保留，程序关闭，信息就丢失。要长期保存，需要把信息写入文件中，保存在硬盘上。

文件有文本文件和二进制文件两种。文本文件保存的是文件所包含字符的 ASCII 码（参看 ASCII 对照表，共包含 128 个字符，除了 95 个可打印的英文字母、数字、标点符号等，还有 33 个控制字符）；二进制文件保存的是文件在内存中的二进制数据形式，内存中是什么样，保存到文件中就是什么样。

文本文件可以直接用“记事本”，“word”等通用的文本编辑软件打开查看，二进制文件只能用对应的读写程序解析后打开查看。

2. 存储在文件中的是文本文件还是二进制文件，是由文件的读写程序决定的，以文本文件方式写入，就是文本文件，同样要以文本文件方式读出；以二进制文件方式写入，就是二进制文件，同样要以二进制文件方式读出。写入和读出的方式不一致，可能看到的的就是乱码。

以文本模式输出时，自动将遇到的换行符“\n”（ASCII 码值为 10）扩充为回车符、换行符两个字符（ASCII 码值为 13 和 10），以文本模式输入时，则执行相反操作。

3. C 语言提供了文件操作方面的库函数，这些库函数的申明包含在 `stdio.h` 头文件中，因此使用这些库函数前要包含该头文件。

4. 文件操作的 3 大步骤：打开文件，读/写文件，关闭文件

5. 为了便于编程时查看输出结果，本次任务先采用文本文件输出方式来保存卡信息（后面实验中再迭代修改为二进制文件方式），每条卡信息占一行，并设计输出的一条卡信息的文本格式为：

卡号##密码##状态##开卡时间##截止时间##累积金额##最后使用时间##使用次数##当前余额##删除标识

卡信息的各成员字段之间用“##”分隔开，其中的时间相关字段以“年-月-日 时：分”字符串格式表示。

## 二. 搭建三层结构

1. 文件操作涉及到数据存储层，程序的三层结构分别对应了界面的输入输出层（`menu.c`），业务流程的处理（`card_service.c`），数据的存取（`card_file.c`）。
2. 添加 `card_file.c` 文件和 `card_file.h` 文件（头文件放对应的函数声明）
3. 在程序的 `AccountManagement` 文件目录下 **建立 data 文件目录（必须手工在 windows 系统的文件管理器中去建立）**

软件 (D:) > AMS > AccountManagement			
名称	修改日期	类型	
data	2021/2/19 15:37	文件夹	
Debug	2021/2/18 17:42	文件夹	
AccountManagement.vcxproj	2021/2/18 17:24	VC++ Project	
AccountManagement.vcxproj.filters	2021/2/18 17:24	VC++ Project Fil...	
AccountManagement.vcxproj.user	2021/2/18 17:14	Visual Studio Pr...	
card_file.c	2018/3/11 20:49	C Source	
card_file.h	2018/3/11 20:49	C/C++ Header	
card_service.c	2021/2/18 17:24	C Source	
card_service.h	2021/2/18 17:24	C/C++ Header	
global.h	2021/2/18 17:24	C/C++ Header	
main.c	2021/2/18 17:44	C Source	
menu.c	2021/2/18 17:44	C Source	
menu.h	2021/2/18 17:24	C/C++ Header	
model.h	2021/2/18 17:24	C/C++ Header	
tool.c	2021/2/18 17:24	C Source	
tool.h	2021/2/18 17:24	C/C++ Header	

在 data 文件目录下手工新建一个文件 card.txt。（提示：也可以不手工建立这个文件，在第一次添加卡信息时，如果没有找到这个文件，会自动建立这个文件，但是可能初次运行时会出现一些“不能打开文件”之类的提示信息，可忽略）

软件 (D:) > AMS > AccountManagement > data			
名称	修改日期	类型	
card.txt	2021/2/19 15:49	文本文档	

注意：在新建“文本文档”的时候只需输入 card 这个文件名，类型扩展名.txt 会自动添加到后面，如果不确定是否添加正确，在“文件资源管理器”菜单上打开“查看”->“显示/隐藏”->“文件扩展名”，这时看到的文件名是 card.txt 正确，是 card.txt.txt 错误，要修改文件名为 card.txt

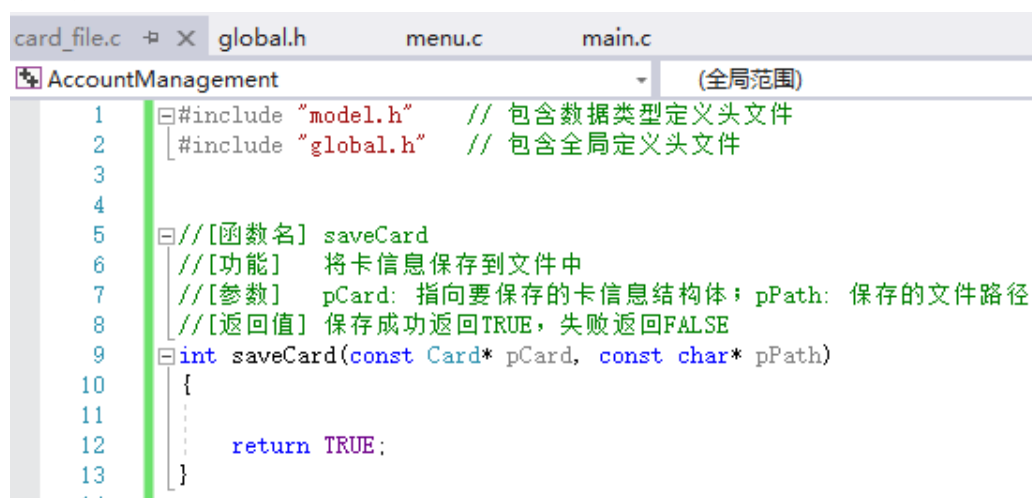
在 global.h 中添加宏定义，定义文件的存取路径，注意定义的是**相对路径**“data\\card.txt”，表示是从当前工程目录（AMS/AccountManagement）下按照此路径可以找到该文件，这样当整个程序文件拷贝复制到其他位置时，不需要修改此处的路径，**不要定义成绝对路径**如 “d:\\AMS\\AccountManagement\\data\\card.txt”

```

1  #pragma once
2
3  #define FALSE 0
4  #define TRUE 1
5
6  #define CARDPATH "data\\card.txt" // 卡信息保存路径
7

```

4. 在 `card_file.c` 中定义 `saveCard` 函数（后面再添加代码，同时在对头文件中添加函数声明）



```

1  #include "model.h" // 包含数据类型定义头文件
2  #include "global.h" // 包含全局定义头文件
3
4
5  //[[函数名] saveCard
6  //[[功能] 将卡信息保存到文件中
7  //[[参数] pCard: 指向要保存的卡信息结构体; pPath: 保存的文件路径
8  //[[返回值] 保存成功返回TRUE, 失败返回FALSE
9  int saveCard(const Card* pCard, const char* pPath)
10 {
11
12     return TRUE;
13 }

```



```

1  #pragma once
2  #include "model.h"
3
4  //函数声明
5  int saveCard(const Card* pCard, const char* pPath);
6

```

5. 在 `card_service.c` 的 `addCard` 函数中调用 `saveCard` 函数，将卡信息保存到文件（前面需 `#include "card_file.h"`）。（去掉函数前面保存到结构体数组中的代码，直接保存到文件中）

```

card_service.c  card_file.h  card_file.c  global.h  menu.c
AccountManagement (全局范围)
6  #include "card_file.h"
7
8  Card aCard[50];    //卡结构体数组
9  int nCount = 0;    //卡结构体数组中的实际卡信息数
10
11  //[[函数名]    addCard
12  //[[功能]      添加卡信息到文件
13  //[[参数]      卡信息结构体
14  //[[返回值]    TRUE: 添加成功; FALSE: 添加失败
15  int addCard(Card crd)
16  {
17  /* if (nCount<50)                //此部分代码删除
18  { //数组未满, 添加一条卡信息
19      aCard[nCount] = crd;
20      //计数增一
21      nCount++;
22      return TRUE; //添加成功
23  }
24  else
25  {
26      printf("数组已满, 不能添加!\n");
27      return FALSE;
28  } */
29  // 将卡信息保存到文件中
30  if (TRUE == saveCard(&crd, CARDPATH))
31  {
32      printf("-----卡信息保存到文件中成功! -----\n");
33      return TRUE;
34  }
35  printf("-----卡信息保存到文件中失败! -----\n");
36  return FALSE;
37  }

```

### 三. 实现卡信息写入文件

C语言使用文件结构体 `FILE` 和文件类型指针 `fp`, 关联指定的文件; 若文件正常打开, 以追加方式写入; 使用 `fprintf` 函数按照下面指定的字符串格式向文件写入卡信息; 最后关闭文件。要使用这些库函数, 前面要 `#include <stdio.h>`

每条写入文件的卡信息结构如下:

卡号##密码##状态##开卡时间##截止时间##累积金额##最后使用时间##使用次数##当前余额##删除标识

各信息字段用##隔开, 其中的时间相关字段要转换为“年一月一日 小时: 分钟”的字符串格式输出, 最后以'\n'结束一条卡信息的输入

在 `card_file.c` 的 `saveCard` 函数中写入如下代码:

```

file.h  card_file.c*  global.h  menu.c  main.c
countManagement  (全局范围)
1  #define _CRT_SECURE_NO_WARNINGS
2  #include<stdio.h>
3
4  #include "model.h"    // 包含数据类型定义头文件
5  #include "global.h"  // 包含全局定义头文件
6  #include "tool.h"
7
8  //[[函数名] saveCard
9  //[[功能]   将卡信息保存到文件中
10 //[[参数]   pCard: 指向要保存的卡信息结构体; pPath: 保存的文件路径
11 //[[返回值] 保存成功返回TRUE, 失败返回FALSE
12 int saveCard(const Card* pCard, const char* pPath)
13 {
14     FILE* fp = NULL;    // 文件结构体指针
15     int isSaveScs; //存储到文件成功否
16     char aStart[30];    //存放转换后的时间字符串
17     char aEnd[30];      //存放转换后的时间字符串
18     char aLast[30];     //存放转换后的时间字符串
19
20     // 以追加的模式打开文件, 如果打开失败, 则以只写的模式打开文件
21     if ((fp = fopen(pPath, "a")) == NULL)
22     {
23         if ((fp = fopen(pPath, "w")) == NULL)
24         {
25             printf("-----添加卡信息打开文件失败! -----\n");
26             return FALSE;
27         }
28     }
29
30     //将time_t类型时间转换为字符串, 字符串格式为"年-月-日 时: 分"
31     timeToString(pCard->tStart, aStart);
32     timeToString(pCard->tEnd, aEnd);
33     timeToString(pCard->tLastTime, aLast);
34     //按指定格式将卡信息输出到文件
35     isSaveScs = fprintf(fp, "%s###s###d###s###s###.1f###s###d###.1f###d\n",
36                         pCard->aName, pCard->aPwd, pCard->nStatus, aStart, aEnd,
37                         pCard->fTotalUse, aLast, pCard->nUseCount, pCard->fBalance, pCard->nDel);
38     if (isSaveScs > 0) {
39         printf("-----卡信息写入文件成功! -----\n");
40         // 关闭文件
41         fclose(fp);
42         return TRUE;
43     }
44     else {
45         printf("-----卡信息写入文件失败! -----\n");
46         // 关闭文件
47         fclose(fp);
48         return FALSE;
49     }
50 }

```

- 思考:
1. C 语言常用的文件打开模式有哪些? 默认的打开模式是什么?
  2. 假如以“追加”模式打开文件失败, 再以“只写”模式打开文件后, 如果原来文件中已有部分卡信息, 这些卡信息是否还存在?
  3. fprintf 函数的使用方法?
  4. If—else 的两个分支中为什么都要 fclose 文件?

#### 四. 编译生成并执行程序

```

D:\AMS\Debug\AccountManagement.exe
请输入卡号(长度为1~18):111
请输入密码(长度为1~8):111
请输入开卡金额(RMB):111

-----*****-----添加的卡信息如下-----*****-----
卡号            密码            状态            开卡金额
111             111             0               111.0

-----****--卡信息写入文件成功!-----****--
-----****--卡信息保存到文件中成功!-----****--
-----*****-----添加卡信息到数组成功!-----*****-----

-----计费系统菜单-----
|
| 1. 添加卡
| 2. 查询卡
| 3. 上机
| 4. 下机
| 5. 充值
| 6. 退费
| 7. 查询统计
| 8. 注销卡
| 0. 退出
|
|-----|

-----
请选择菜单项编号(0~8):1

-----****--添加卡-----****--
请输入卡号(长度为1~18):123
请输入密码(长度为1~8):123
请输入开卡金额(RMB):123

-----*****-----添加的卡信息如下-----*****-----
卡号            密码            状态            开卡金额
123             123             0               123.0

-----****--卡信息写入文件成功!-----****--
-----****--卡信息保存到文件中成功!-----****--
-----*****-----添加卡信息到数组成功!-----*****-----

```

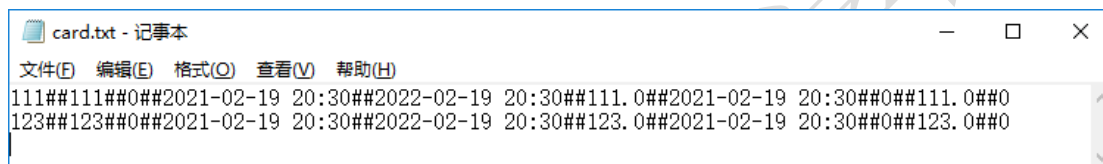
修改 menu.c 中 add 函数中的提示内容：

```

98 //卡信息添加到文件中
99 if (FALSE == addCard(card))
100 {
101     printf("-----*****-----添加卡信息到文件失败! -----*****-----\n");
102 }
103 else
104 {
105     printf("-----*****-----添加卡信息到文件成功! -----*****-----\n");
106 }

```

按照程序文件目录 “AccountManagement/data/card.txt” 在 “文件资源管理器” 找到 card.txt 文本文件，用 “记事本” 打开，可看到我们添加的 2 条卡信息（注意在 “记事本” 中尽管我们看到一行的末尾还有很多空白，但实际上后面已经没有字符了，当然也没有空格字符在后面，是因为 “记事本” 程序对卡信息数据最后的回车换行符的处理，使得后面的一条卡信息数据转到了下一行开始显示）



## 五. 读取文件中的卡信息

程序启动后，将文件中的卡信息读出，存放于结构体数组中；查询卡信息时，从结构体数组中查找。

1. 在 card\_file.c 中定义 readCard 函数（头文件中加函数声明），实现从文件读取卡信息。用文件结构体 FILE 和文件类型指针 fp，关联指定的文件，以只读方式读取；若文件正常打开，从文件读取卡信息；最后关闭。
2. 打开文件后，使用 fgets 函数每次从文件中读取一行即一条卡信息，使用 feof 函数判断是否到文件末尾，结束循环。

代码如下：

```

file.h  card_file.c  menu.c  main.c
countManagement  (全局范围)
52  #define CARDCHARNUM 256 //从卡信息文件中读取的字符数
53  /******
54  [函数名] readCard
55  [功能] 从文件中读取卡信息到结构体数组
56  [参数] pCard: 指向结构体数组; pPath: 文件路径
57  [返回值] 正确读取返回TRUE, 否则返回FALSE
58  *****/
59  int readCard(Card* pCard, const char* pPath)
60  {
61      FILE* fp = NULL; // 文件结构体指针
62      int nIndex = 0; // 卡信息数量
63      char aBuf[CARDCHARNUM] = { 0 }; // 保存从文件中读取的数据
64
65      int i = 0; // 结构体数组下标
66
67      // 以只读的方式打开文件
68      if ((fp = fopen(pPath, "r")) == NULL)
69      {
70          printf("----*---读取卡信息打开文件失败! ----*---");
71          return FALSE;
72      }
73
74      // 从文件中逐行读取卡信息
75      while (!feof(fp))
76      {
77          memset(aBuf, 0, CARDCHARNUM); // 清空字符数组
78          if (fgets(aBuf, CARDCHARNUM, fp) != NULL) // 从文件中每次读取一行到aBuf
79          {
80              if (strlen(aBuf) != 0)
81              { // 将一条卡信息的各分里解析后存放到结构体
82                  pCard[i] = praseCard(aBuf);
83              }
84              i++;
85          }
86      }
87      // 关闭文件
88      fclose(fp);
89      return TRUE;

```

其中用到 `memset` 函数, 原型为: `void* memset (void* buffer, int c, int count);`

`Buffer`: 为指针或是数组, `c`: 是赋给 `buffer` 的值, `count`: 是 `buffer` 的长度。  
将 `buffer` 所指向的 `count` 大小的块内存中的每个字节的内容全部设置为 `c` 指定的 ASCII 值, 这个函数通常为新申请的内存做初始化工作或清空。

其中用到 `strlen` 函数, 其原型为: `unsigned int strlen (char *s);`  
`s` 为指定的字符串。用来计算指定的字符串 `s` 的长度, 不包括结束符“\0”。  
返回字符串 `s` 的字符数。

使用这两个函数需包含头文件: `#include<string.h>`

3. 从文件中读取的一行卡信息是“##”分隔的字符串, 放在 `aBuf` 变量中, 字符串样式为:

```
111##111##0##2021-02-19 20:30##2022-02-19 20:30##111.0##2021-02-19 20:30##0##111.0##0
```

需要分解出每个卡信息字段, 分别存放到结构体的对应成员分量中 (编写 `praseCard` 函数实现)。



思考：1. 了解 feof 函数？

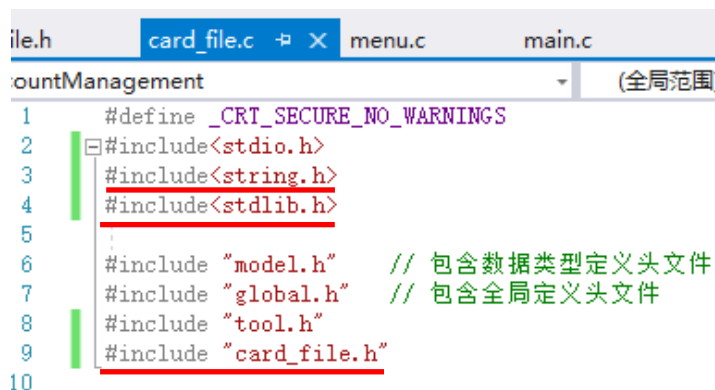
2. 读取文件信息前，为什么要清空 aBuf 中字符？

3. 了解 fgets 函数，什么情况下读取停止？第二个参数 CARDCHARNUM 起什么作用？

4. 文件输入与标准输入的相同和区别？

## 六. 解析卡信息

在 card\_file.c 中定义 praseCard 函数（card\_file.h 头文件中加函数声明，card\_file.c 文件前面添加#include "card\_file.h"），用来实现卡信息字段的分解。



```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include<stdio.h>
3  #include<string.h>
4  #include<stdlib.h>
5
6  #include "model.h"    // 包含数据类型定义头文件
7  #include "global.h"  // 包含全局定义头文件
8  #include "tool.h"
9  #include "card_file.h"
10

```

1. 其中会用到 strtok 函数，按指定的分割字符来分解字符串。函数原型：

Char \*strtok (char \*s,char \*delim) ;

参数 s 是要分解的字符串，参数 delim 是分割字符，比如这里的“##”，返回指向分解出来的字符的指针

解析出来的字符串分量先放在 flag 数组中，flag[10][20]是二维字符数组，第一维 10 是因为“##”分隔的 10 个分量，共 10 个字符串，第二维 20 是每个字符串的最大长度；所以每个 flag[i]是一个字符数组，存放解析后的一个字符串，比如上面例子解析后 flag[0]中是“111”，flag[3]中是“2021-02-19 20:30”，flag[5]中是“111.0”。。。

2. 解析出来的卡信息分量都是字符串类型，要转换成合适的类型再存放到卡信息结构体中；其中用到 atoi 函数，atof 函数（需#include<stdlib.h>），这两个函数分别将字符串类型转换成整数和浮点数类型；

3. 解析出来的时间分量是“年—月—日 小时：分钟”这样的字符串格式，调用之前编写过的 stringToTime 函数转换成日历时间，再存放到结构体中

代码如下：

```

_file.h  card_file.c*  menu.c  main.c
ccountManagement  (全局范围)
93  // [函数名] praseCard
94  // [功能] 将一条卡信息字符串解析后放入一个卡结构体
95  // [参数] 卡信息字符串
96  // [返回值] 卡结构体
97  Card praseCard(char* pBuf)
98  {
99      Card card;
100      char flag[10][20]; //临时存放分量
101      char* str; //每次解析出来的字符串
102      char* buf;
103      int index = 0; //数组下标
104      buf = pBuf; //第一次调用strtok函数时, buf为解析字符串
105      while ((str = strtok(buf, "##")) != NULL)
106      {
107          strcpy(flag[index], str);
108          buf = NULL; //后面调用strtok函数时, 第一个参数为NULL
109          index++;
110      }
111      // 分割后的内容保存到结构体中
112      strcpy(card.aName, flag[0]);
113      strcpy(card.aPwd, flag[1]);
114      card.nStatus = atoi(flag[2]);
115      card.tStart = stringToTime(flag[3]);
116      card.tEnd = stringToTime(flag[4]);
117      card.fTotalUse = atof(flag[5]);
118      card.tLastTime = stringToTime(flag[6]);
119      card.nUseCount = atoi(flag[7]);
120      card.fBalance = atof(flag[8]);
121      card.nDel = atoi(flag[9]);
122      //返回卡信息结构体
123      return card;
124  }

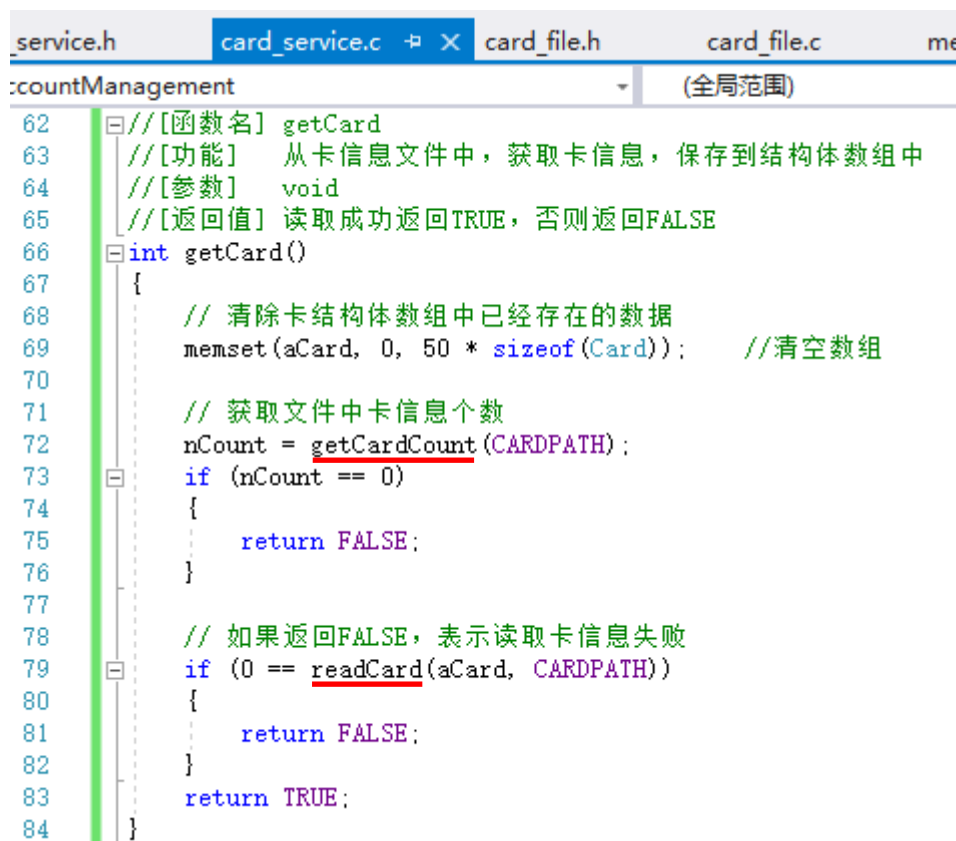
```

七. 将解析的卡信息保存到结构体数组中

1. 在 card\_service.c 文件中定义 getCard 函数（对应头文件中添加函数声明）

为保证查询卡时，文件和结构体数组中的数据一致，先清空结构体数组中数据，然后再从文件中读取所有卡信息到结构体数组。

getCard 函数调用 getCardCount 函数得到其返回值----卡文件中的卡信息数量，通过判断返回值进行不同处理：返回值为 0，表示读取信息失败或文件中没有卡信息；返回其他非零值，就进一步读取卡信息。



```

service.h  card_service.c  card_file.h  card_file.c  me
countManagement (全局范围)
62  // [函数名] getCard
63  // [功能] 从卡信息文件中，获取卡信息，保存到结构体数组中
64  // [参数] void
65  // [返回值] 读取成功返回TRUE，否则返回FALSE
66  int getCard()
67  {
68      // 清除卡结构体数组中已经存在的数据
69      memset(aCard, 0, 50 * sizeof(Card)); // 清空数组
70
71      // 获取文件中卡信息个数
72      nCount = getCardCount(CARDPATH);
73      if (nCount == 0)
74      {
75          return FALSE;
76      }
77
78      // 如果返回FALSE，表示读取卡信息失败
79      if (0 == readCard(aCard, CARDPATH))
80      {
81          return FALSE;
82      }
83      return TRUE;
84  }

```

注意存在的问题：

卡信息结构体数组 `aCard[50]` 最多只能存放 50 条卡信息，这里程序并没有考虑卡信息数量大于 50 的情况，是因为后面我们的迭代要丢弃结构体数组，使用链表动态存取卡信息，不再受程序中卡信息数量变化的影响。

## 2. 在 `card_file.c` 中定义 `getCardCount` 函数（对应头文件添加函数声明）

`getCardCount` 函数代码类似于 `readCard` 函数，只是在读取到一行卡信息后进行计数

```

_service.c  card_file.c  menu.c  main.c
ccountManagement (全局范围)
126 // [函数名] getCardCount
127 // [功能] 获取卡信息文件中，卡信息数量
128 // [参数] 文件路径
129 // [返回值] 卡信息数量
130 int getCardCount(const char* pPath)
131 {
132     FILE* fp = NULL; // 文件指针
133     int nIndex = 0; // 卡信息数量
134     char aBuf[CARDCHARNUM] = { 0 }; // 保存从文件中读取的数据
135     // 以只读模式打开文件
136     if ((fp = fopen(pPath, "r")) == NULL)
137     {
138         printf("-----获取卡信息数量打开文件失败！-----");
139         return 0;
140     }
141     // 逐行读取文件内容，获取的文件行数就是卡信息数
142     while (!feof(fp))
143     {
144         memset(aBuf, 0, CARDCHARNUM); // 清空字符数组
145         fgets(aBuf, CARDCHARNUM, fp); // 从文件中每次读取一行到aBuf
146         if (strlen(aBuf) != 0) nIndex++;
147     }
148     // 关闭文件
149     fclose(fp);
150     return nIndex;
151 }

```

#### 八. 查询并显示

在 card\_service.c 文件中修改 queryCard 函数，调用 getCard 函数后，再到结构体数组中查询（前面需#include "card\_service.h"）

```

card_service.c  card_file.c  menu.c  main.c
AccountManagement (全局范围)
45 Card* queryCard(const char* pName)
46 {
47     Card* pCard = NULL;
48     int i;
49
50     // 如果从卡信息文件中读取卡信息失败，则返回NULL
51     if (FALSE == getCard())
52     {
53         return NULL;
54     }
55     // 在结构体数组中查找指定卡号
56     for (i = 0; i < nCount; i++)
57     {
58         if (strcmp(aCard[i].aName, pName) == 0)
59         { // 在结构体数组中找到，返回卡信息地址
60             pCard = &aCard[i];
61             return pCard;
62         }
63     }
64     // 没有找到，返回NULL
65     return pCard;
66 }

```

## 九. 编译运行程序

添加到文件的卡信息

```
card.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
111##111##0##2021-02-19 20:30##2022-02-19 20:30##111.0##2021-02-19 20:30##0##111.0##0
123##123##0##2021-02-19 20:30##2022-02-19 20:30##123.0##2021-02-19 20:30##0##123.0##0
121##121##0##2021-02-19 21:00##2022-02-19 21:00##121.0##2021-02-19 21:00##0##121.0##0
czyh##czyh##0##2021-02-20 19:49##2022-02-20 19:49##1000.0##2021-02-20 19:49##0##1000.0##0
zhy##zhy##0##2021-02-20 19:49##2022-02-20 19:49##800.0##2021-02-20 19:49##0##800.0##0
```

查询到的卡信息

```
D:\AMS\Debug\AccountManagement.exe
请选择菜单项编号 (0 8): 2
-----***-----查询卡-----***-----
请输入要查询的卡号(长度为1~18):121
-----*****-----查到的卡信息如下-----*****-----
卡号      状态      余额      累计金额      使用次数      上次使用时间
121       0         121.0     121.0         0             2021-02-19 21:00
-----*****-----*****-----*****-----
-----计费系统菜单-----
|| 1. 添加卡 ||
|| 2. 查询卡 ||
|| 3. 上机   ||
|| 4. 下机   ||
|| 5. 充值   ||
|| 6. 退费   ||
|| 7. 查询统计 ||
|| 8. 注销卡 ||
|| 0. 退出   ||
||          ||
-----
请选择菜单项编号 (0 8): 2
-----***-----查询卡-----***-----
请输入要查询的卡号(长度为1~18):czyh
-----*****-----查到的卡信息如下-----*****-----
卡号      状态      余额      累计金额      使用次数      上次使用时间
czyh      0         1000.0    1000.0         0             2021-02-20 19:49
-----*****-----*****-----*****-----
-----计费系统菜单-----
|| 1. 添加卡 ||
```

## 十. 总结

本次任务的层次结构和主要调用关系

