

一. 文件基础操作

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。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，二进制文件保存的是文件在内存中的二进制数据形式。

文本文件可以直接用“记事本”，“word”等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。否则可能就是乱码。

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。(C++ 中的 endl 相当于换行符“\n”加上刷新缓冲区的作用)

3. C++ 提供文件流类用于文件读写，使用时需包含头文件 `#include <fstream>`

`fstream` (读写文件)、`ifstream` (读文件)、`ofstream` (写文件)

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，并设计输出的文本格式为：

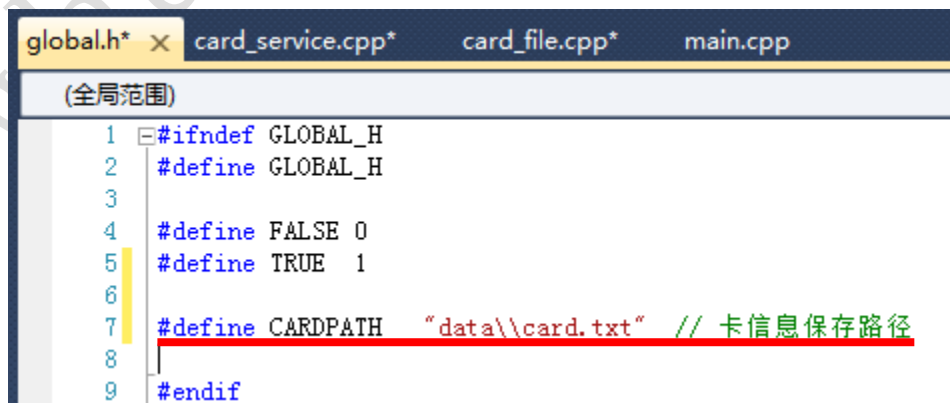
卡号##密码##状态##开卡时间##截止时间##累积金额##最后使用时间##使用次数##当前余额##删除标识

各信息字段用##隔开，其中的时间相关字段要转换为“年—月—日 小时：分钟”的字符串格式输出。

二. 搭建三层结构

1. 程序的三层结构分别对应了界面的输入输出层 (menu.cpp), 业务流程的处理 (card_service.cpp), 数据的存取 (card_file.cpp)。
2. 添加 card_file.cpp 文件和 card_file.h 文件 (头文件放函数声明)
3. 在程序的 AccountManagement 目录下 **建立 data 目录 (必须手工建立)**，在 global.h 中添加宏定义，定义文件的存取路径

在 data 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会自动建立这个文件，但是可能会出来一些“不能打开文件”之类的提示信息，可忽略)



4. 在 card_file.cpp 中定义 saveCard 函数 (后面再添加代码)

```

card_file.cpp* x main.cpp
(全局范围)
1 #include "model.h"
2 #include "global.h"
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     return TRUE;
12 }

```

5. 在 card_service.cpp 的 addCard 函数中调用 saveCard 函数, 将卡信息保存到文件 (需 #include "card_file.h")。 (注释掉前面保存到结构体数组中的代码, 待程序编译运行通过后删除)

```

file.cpp card_service.cpp* x
(全局范围)
16 int addCard(Card crd)
17 {
18     /* if(nCount<50) //注释掉, 待删除
19     { //数组未滿, 添加一条卡信息
20         strcpy(aCard[nCount].aName, crd.aName);
21         strcpy(aCard[nCount].aPwd, crd.aPwd);
22         aCard[nCount].nStatus=crd.nStatus;
23         aCard[nCount].tStart=crd.tStart;
24         aCard[nCount].tEnd=crd.tEnd;
25         aCard[nCount].fTotalUse=crd.fTotalUse;
26         aCard[nCount].tLastTime=crd.tLastTime;
27         aCard[nCount].nUseCount=crd.nUseCount;
28         aCard[nCount].fBalance=crd.fBalance;
29         aCard[nCount].nDel=crd.nDel;
30         //计数增一
31         nCount++;
32         return TRUE;
33     }
34     else
35     {
36         cout<<"数组已滿, 不能添加!"<<endl;
37         return FALSE;
38     }
39     // 将卡信息保存到文件中
40     if(TRUE == saveCard(&crd, CARDPATH))
41     {
42         return TRUE;
43     }
44     return FALSE;
45 }

```

三. 实现卡信息写入文件

创建一个输出文件流 `ofstream` 对象管理输出流, 关联指定文件, 以追加方式写入; 若文件正常打开, 使用流输出方式向文件写入卡信息; 最后关闭。

使用“<<”和“>>”运算符向文件中写或读基本数据类型的数据时，是以文本方式操作文件的，即将要写或读的数据直接当做字符串存取自文件中。

其中使用 ofstream，需 #include <fstream>

using namespace std;

```

card_file.cpp* x card_service.cpp
(全局范围)
1 #include <iostream>
2 #include <fstream>
3
4 #include "model.h"
5 #include "global.h"
6 #include "tool.h"
7
8 using namespace std;
9
10 //【函数名】 saveCard
11 //【功能】 将卡信息保存到文件中
12 //【参数】 pCard: 指向要保存的卡信息结构体; pPath: 保存的文件路径
13 //【返回值】 保存成功返回TRUE，失败返回FALSE
14 int saveCard(const Card* pCard, const char* pPath)
15 {
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     timeToString(pCard->tStart, aStart);
21     timeToString(pCard->tEnd, aEnd);
22     timeToString(pCard->tLastTime, aLast);
23
24     ofstream ofile(pPath, ios_base::out | ios_base::app); //以追加方式写入
25     if(!ofile.is_open())
26     {
27         cerr<<"文件无法正确打开！不能写入卡信息！"<<endl;
28         exit(EXIT_FAILURE);
29     }
30     //向文件写入卡信息
31     ofile<<pCard->aName<<"##"<<pCard->aPwd<<"##"<<pCard->nStatus<<"##";
32     ofile<<aStart<<"##"<<aEnd<<"##"<<pCard->fTotalUse<<"##"<<aLast<<"##";
33     ofile<<pCard->nUseCount<<"##"<<pCard->fBalance<<"##"<<pCard->nDel<<endl;
34     ofile.close();
35
36     cout<<endl;
37     cout<<"-----*****-----卡信息成功存入文件!-----*****-----"<<endl<<endl;
38     return TRUE;
39 }

```

- 思考：1. C++常用的文件打开模式有哪些？默认的打开模式是什么？
2. 了解 is_open 方法（检查打开文件是否成功，文件模式是否合适），还有其他类似常用方法？

四．编译生成并执行程序

The screenshot shows a Windows application window titled "E:\AMS\Debug\AccountManagement.exe". The application is a text-based menu system for account management. It displays two successful card additions, each highlighted with a red rectangular box. The first addition uses card number "test1", password "123", and amount "123". The second addition uses card number "test2", password "111", and amount "111". Both additions show a confirmation message: "卡信息成功存入文件!" and "添加卡信息成功!".

```
E:\AMS\Debug\AccountManagement.exe

-----添加卡-----
请输入卡号（长度为1~18）:test1
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
      test1      123      0      123.00

-----*****卡信息成功存入文件!*****-----
-----*****添加卡信息成功!*****-----

-----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        ::
::

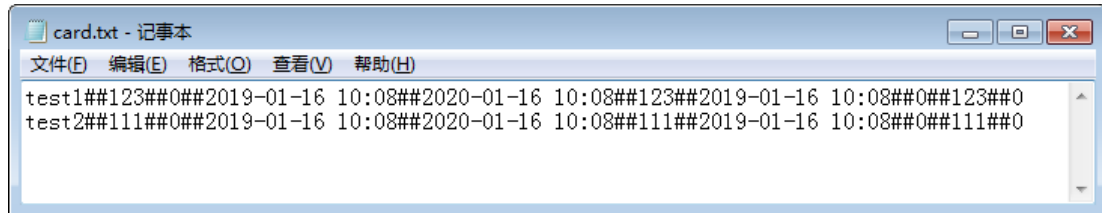
请选择菜单项编号（0~8）: 1

-----添加卡-----
请输入卡号（长度为1~18）:test2
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
      test2      111      0      111.00

-----*****卡信息成功存入文件!*****-----
-----*****添加卡信息成功!*****-----
```

用“记事本”打开程序目录“AccountManagement/data/card.txt”文件，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，是由于“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，存放到结构体数组中；查询卡信息时，从结构体数组中查找。

1. 在 `card_file.cpp` 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。创建一个输入文件流 `ifstream` 对象管理输入流，关联指定文件，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，遍历文件，每次从文件中读取一行，使用 `eof` 方法判断是否到文件末尾，结束循环。
3. 读取的一行卡信息是“##”分隔的字符串，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）。

代码如下：

```

card_file.cpp* x card_service.cpp
全局范围)
42 #define CARDCHARNUM 256 //从卡信息文件中读取的字符数最大值
43 /*****
44 [函数名] readCard
45 [功能] 从文件中读取卡信息到结构体数组
46 [参数] pCard: 指向结构体数组; pPath: 文件路径
47 [返回值] 正确读取返回TRUE, 否则返回FALSE
48 *****/
49 int readCard(Card* pCard, const char* pPath)
50 {
51     int nIndex = 0; // 卡信息数量
52     char aBuf[CARDCHARNUM] = {0}; // 保存从文件中读取的一行数据
53     int i=0; // 结构体数组下标
54
55     ifstream ifile;
56     // 默认以只读的方式打开文件
57     ifile.open(pPath);
58     if(ifile.is_open())
59     { // 遍历文件
60         while(!ifile.eof())
61         {
62             memset(aBuf, 0, CARDCHARNUM); // 清空字符数组
63             ifile.getline(aBuf, CARDCHARNUM); // 读取一行卡信息
64             if(strlen(aBuf) != 0)
65             { // 将一条卡信息的各分里解析后存放到结构体
66                 pCard[i] = praseCard(aBuf);
67             }
68             i++;
69         }
70     }
71     else{
72         cout<<"文件无法正确打开! 不能读取卡信息!"<<endl;
73         ifile.close();
74         return FALSE;
75     }
76     // 关闭文件
77     ifile.close();
78     return TRUE;
79 }

```

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，不包括结束字符 `"\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>`

- 思考: 1. 了解 `eof` 方法以及流状态?
 2. 读取文件信息前, 为什么要清空 `aBuf` 中字符?

3. 了解 `getline` 方法，什么情况下读取停止？第二个参数 `CARDCHARNUM` 起什么作用？
4. 文件输入输出与标准输入输出的相同和区别？

六. 解析卡信息

在 `card_file.cpp` 中定义 `praseCard` 函数（`card_file.h` 头文件中加函数声明，`card_file.cpp` 文件头添加 `#include "card_file.h"`），实现卡信息字段的分解。

其中用到 `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，

返回指向分解出来的字符的指针

其中用到 `atoi` 函数，`atof` 函数（需 `#include <stdlib.h>`），分别将字符串转换成整数和浮点数

解析出来的时间分量是字符串格式，调用之前编写的 `stringToTime` 函数转换成日历时间，再存放到结构体

代码如下：

```

d_file.h  card_file.cpp* x  card_service.cpp
[全局范围]
84  // [函数名] praseCard
85  // [功能]  将一条卡信息字符串解析后放入一个卡结构体
86  // [参数]  卡信息字符串
87  // [返回值] 卡结构体
88  Card praseCard(char* pBuf)
89  {
90      Card card;
91      char flag[10][20]; //临时存放分里
92      char* str; //每次解析出来的字符串
93      char* buf;
94      int index=0; //数组下标
95
96      buf=pBuf; //第一次调用strtok函数时，buf为解析字符串
97      while((str=strtok(buf, "##")) != NULL)
98      {
99          strcpy(flag[index], str);
100          buf=NULL; //后面调用strtok函数时，第一个参数为NULL
101          index++;
102      }
103      // 分割后的内容保存到结构体中
104      strcpy(card.aName, flag[0]);
105      strcpy(card.aPwd, flag[1]);
106      card.nStatus=atoi(flag[2]);
107      card.tStart=stringToTime(flag[3]);
108      card.tEnd=stringToTime(flag[4]);
109      card.fTotalUse=atof(flag[5]);
110      card.tLastTime=stringToTime(flag[6]);
111      card.nUseCount=atoi(flag[7]);
112      card.fBalance=atof(flag[8]);
113      card.nDel=atoi(flag[9]);
114      //返回卡信息结构体
115      return card;
116  }
117

```

七. 将解析的卡信息保存到结构体数组中

1. 在 card_service.cpp 文件中定义 getCard 函数（对应头文件添加函数声明）

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；

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


```

card_service.cpp x card_file.cpp
(全局范围)
76 //[[函数名] getCard
77 //[[功能] 从卡信息文件中，获取卡信息，保存到结构体数组中
78 //[[参数] void
79 //[[返回值] 读取成功返回TRUE，否则返回FALSE
80 int getCard()
81 {
82     // 清除卡结构体数组中已经存在的数据
83     memset(aCard, 0, 50*sizeof(Card)); //清空数组
84
85     // 获取文件中卡信息个数
86     nCount = getCardCount(CARDPATH);
87     if(nCount == 0)
88     {
89         return FALSE;
90     }
91     else if(nCount == -1)
92     {
93         cout<<"文件无法打开！"<<endl;
94         return FALSE;
95     }
96     // 如果返回FALSE，表示读取卡信息失败
97     if(0 == readCard(aCard, CARDPATH))
98     {
99         return FALSE;
100     }
101     return TRUE;
102 }

```

```

d_service.cpp card_file.cpp x
(全局范围)
119 //[[函数名] getCardCount
120 //[[功能] 获取卡信息文件中，卡信息数量
121 //[[参数] 文件路径
122 //[[返回值] 卡信息数量
123 int getCardCount(const char* pPath)
124 {
125     int nIndex = 0; // 卡信息数量
126     char aBuf[CARDCHARNUM]={0}; //保存从文件中读取的数据
127
128     ifstream ifile;
129     // 默认以只读的方式打开文件
130     ifile.open(pPath);
131     if(ifile.is_open())
132     { //遍历文件
133         while(!ifile.eof())
134         {
135             memset(aBuf, 0, CARDCHARNUM); //清空字符数组
136             ifile.getline(aBuf, CARDCHARNUM); //读取一行卡信息
137             if(strlen(aBuf) != 0) nIndex++;
138         }
139     }
140     else{
141         cout<<"文件无法正确打开！不能读取卡信息！"<<endl;
142         ifile.close();
143         return -1;
144     }
145     // 关闭文件
146     ifile.close();
147     return nIndex;
148 }

```

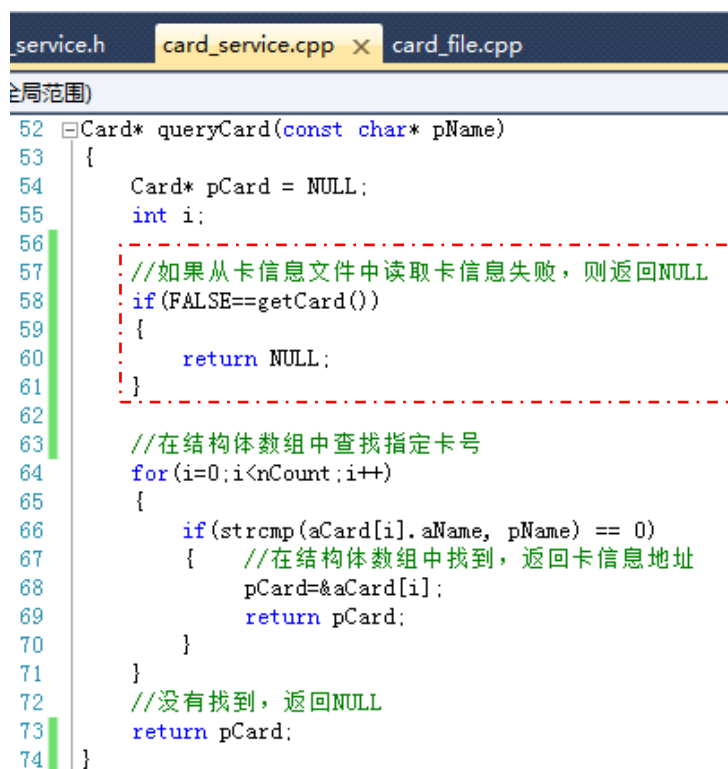
getCardCount 函数中当卡信息文件没有正常打开时，设置了返回值为-1，区别与返回值

nIndex 的其他情况，getCard 函数调用 getCardCount 函数得到其返回值，通过判断返回值进行不同处理：返回值为-1，表示文件没有正确打开；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。（可以自行在 global.h 文件中定义一个宏，比如：#define FILENOTOPEN -1；程序代码修改为 return FILENOTOPEN；）

思考：getCardCount 中 else 分支中为什么要 close 文件？

八. 查询并显示

在 card_service.cpp 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"）



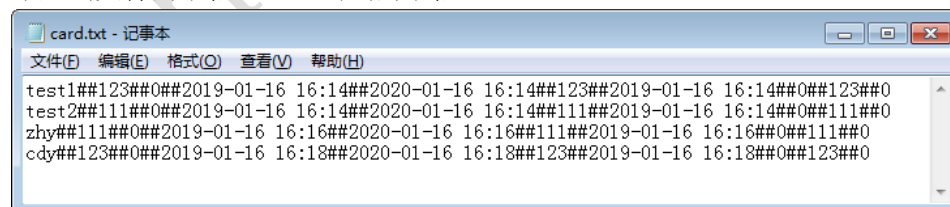
```

52 Card* queryCard(const char* pName)
53 {
54     Card* pCard = NULL;
55     int i;
56
57     //如果从卡信息文件中读取卡信息失败，则返回NULL
58     if(FALSE==getCard())
59     {
60         return NULL;
61     }
62
63     //在结构体数组中查找指定卡号
64     for(i=0;i<nCount;i++)
65     {
66         if(strcmp(aCard[i].aName, pName) == 0)
67         { //在结构体数组中找到，返回卡信息地址
68             pCard=&aCard[i];
69             return pCard;
70         }
71     }
72     //没有找到，返回NULL
73     return pCard;
74 }

```

九. 编译运行程序

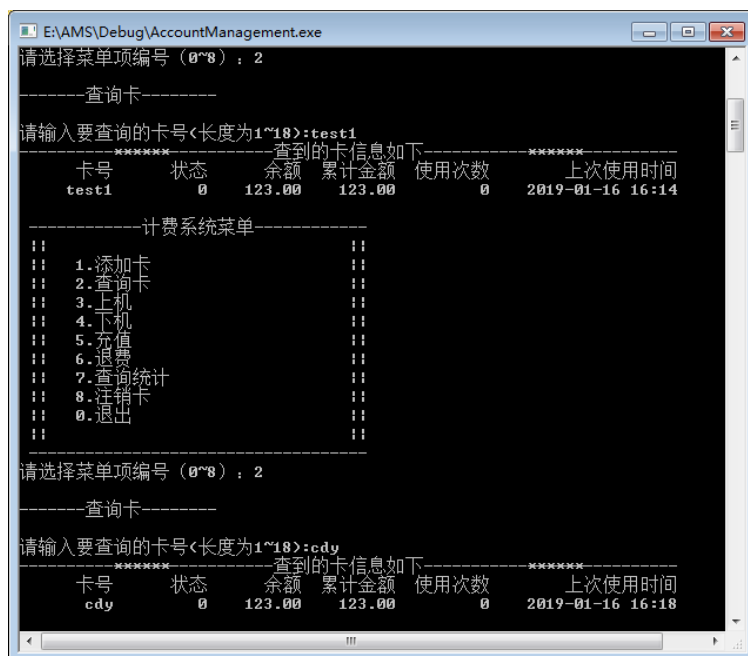
添加到文件的卡信息，查询到的卡信息



```

test1##123##0##2019-01-16 16:14##2020-01-16 16:14##123##2019-01-16 16:14##0##123##0
test2##111##0##2019-01-16 16:14##2020-01-16 16:14##111##2019-01-16 16:14##0##111##0
zhy##111##0##2019-01-16 16:16##2020-01-16 16:16##111##2019-01-16 16:16##0##111##0
cdy##123##0##2019-01-16 16:18##2020-01-16 16:18##123##2019-01-16 16:18##0##123##0

```



十. 总结

本次任务的层次结构和主要调用关系

