

一. 更新上机卡

对已添加的卡,当用户通过输入相匹配的卡号和密码,开始上机,在界面显示上机信息,更新上机状态。

1. 查找上机卡

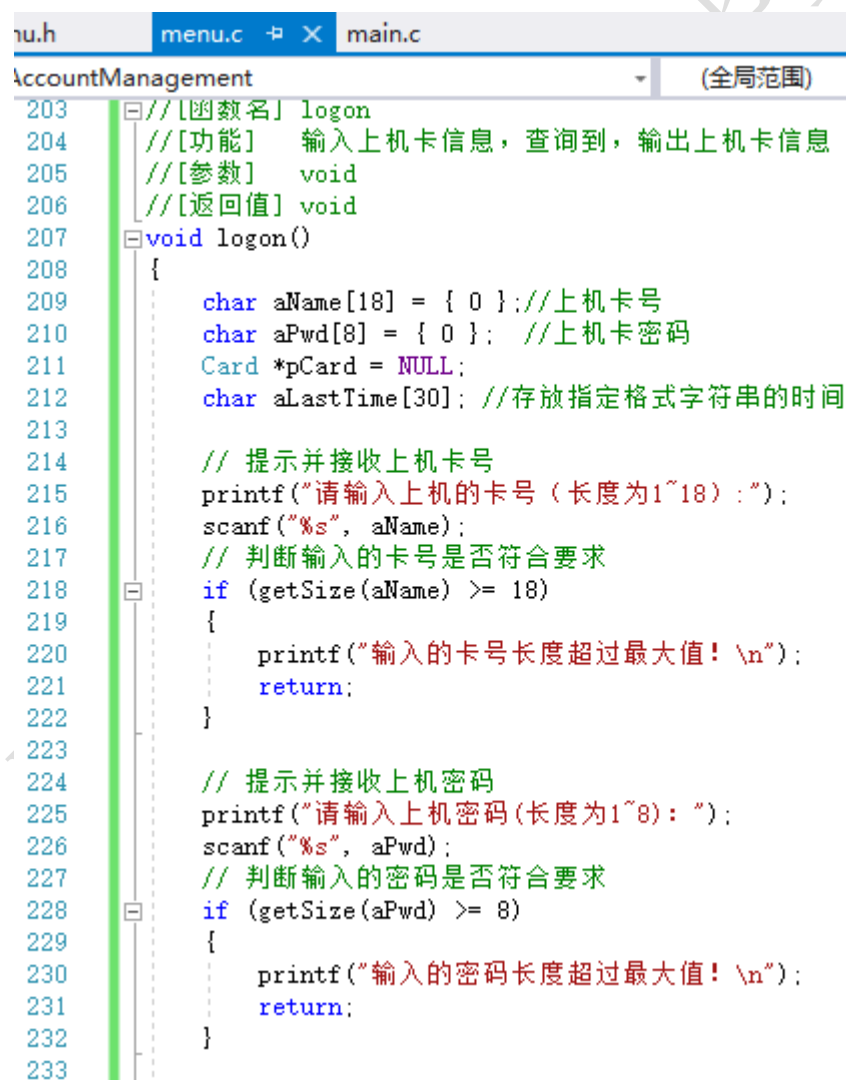
在 main.c 的 main 函数中,用户选择“3.上机”时,调用 logon 函数,实现上机操作。修改代码如下:

```

37         case 3:
38             printf("\n-----***-----上机-----***-----\n\n");
39             logon();
40             break;

```

在 menu.c 中创建 logon 函数(同时在对头文件中添加函数声明),提示并接收用户输入的卡号和密码,获取上机的卡信息结果后,输出不同信息。如果未获得卡信息,表示上机失败,提示用户“上机失败”,否则,通过卡结构体指针,向界面列表输出卡号,余额和上机时间,上机时间默认为卡的最后使用时间,转换成字符串格式输出。代码如下:



```

menu.h  menu.c  main.c
AccountManagement (全局范围)
203 // [函数名] logon
204 // [功能] 输入上机卡信息, 查询到, 输出上机卡信息
205 // [参数] void
206 // [返回值] void
207 void logon()
208 {
209     char aName[18] = { 0 }; // 上机卡号
210     char aPwd[8] = { 0 }; // 上机卡密码
211     Card *pCard = NULL;
212     char aLastTime[30]; // 存放指定格式字符串的时间
213
214     // 提示并接收上机卡号
215     printf("请输入上机的卡号 (长度为1~18): ");
216     scanf("%s", aName);
217     // 判断输入的卡号是否符合要求
218     if (getSize(aName) >= 18)
219     {
220         printf("输入的卡号长度超过最大值! \n");
221         return;
222     }
223
224     // 提示并接收上机密码
225     printf("请输入上机密码 (长度为1~8): ");
226     scanf("%s", aPwd);
227     // 判断输入的密码是否符合要求
228     if (getSize(aPwd) >= 8)
229     {
230         printf("输入的密码长度超过最大值! \n");
231         return;
232     }
233

```

```

234 //开始上机，获取上机结果
235 pCard = doLogon(aName, aPwd);
236
237 //根据上机结果，提示不同信息
238 if (pCard == NULL)
239 {
240     printf("-----上机失败! -----\n");
241 }
242 else
243 {
244     printf("-----上机的卡信息如下-----\n");
245     // 输出表格的表头
246     printf("卡号\t余额\t上机时间\n");
247     //将time_t类型时间转换为字符串，字符串格式为“年-月-日 时：分”
248     timeToString(pCard->tLastTime, aLastTime);
249     //输出查到的卡信息
250     printf("%s\t%.1f\t%s\n", pCard->aName, pCard->fBalance, aLastTime);
251 }
252 }

```

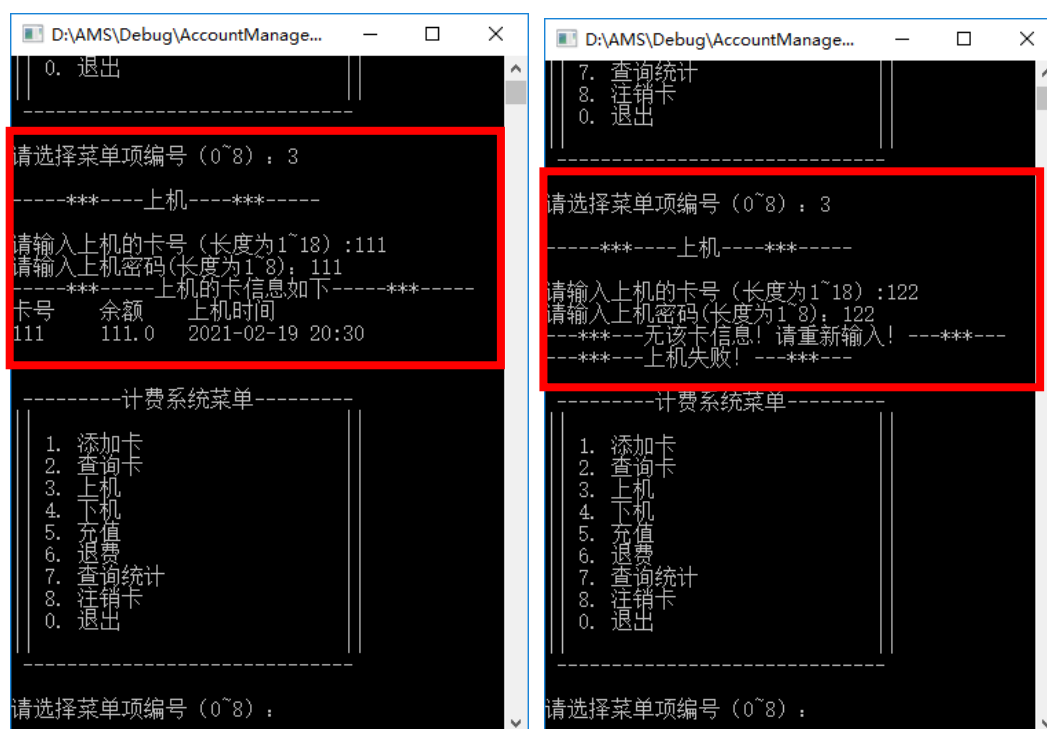
在 card_service.c 中定义 doLogon 函数（同时在对头文件中添加函数声明），查找上机卡的卡信息。先将文件中卡信息读取到链表，然后遍历链表查找匹配的卡信息，代码如下：

```

service.h  card_service.c  menu.h  menu.c  main.c
ccountManagement (全局范围)
224 //函数名 doLogon
225 //功能 从文件读取卡信息到链表，在链表中查询匹配的卡信息
226 //参数 pName: 上机卡号; pPwd: 上机密码
227 //返回值 上机卡信息结构体
228 Card* doLogon(const char* pName, const char* pPwd)
229 {
230     lpCardNode node = NULL; //当前节点
231
232     //从卡信息文件中读取卡信息到链表，失败返回NULL
233     if (FALSE == getCard())
234     {
235         return NULL;
236     }
237     //当前节点指向第一个结点
238     if (cardList != NULL) node = cardList->next;
239
240     while (node != NULL)
241     {
242         //在链表中查找是否有对应的卡信息
243         if ((strcmp(node->data.aName, pName) == 0) && (strcmp(node->data.aPwd, pPwd) == 0))
244         {
245             return &node->data;
246         }
247         node = node->next;
248     }
249     if (node == NULL) printf("-----无该卡信息! 请重新输入! -----\n");
250     return NULL;
251 }
252

```

编译连接并运行程序：



2. 更新卡信息

查询到上机卡信息后，卡的一些状态信息需要修改，要更新链表中的卡信息，同时更新文件中的卡信息（调用 `updateCard` 函数来实现），更新成功才表示上机成功，再返回该卡的卡信息地址。修改 `doLogon` 函数如下：

```

d_service.h  card_service.c*  menu.h  menu.c  main.c
AccountManagement  (全局范围)  dol
224  //函数名 doLogon
225  //功能 从文件读取卡信息到链表，在链表中查询匹配的卡信息，更新链表和文件中对应卡信息
226  //参数  pName: 上机卡号; pPwd: 上机密码
227  //返回值 上机卡信息结构体
228  Card* doLogon(const char* pName, const char* pPwd)
229  {
230      lpCardNode node = NULL; //当前节点
231      int nIndex = 0; //要查找节点在链表中序号，用于更新卡信息
232
233      //从卡信息文件中读取卡信息到链表，失败返回NULL
234      if (FALSE == getCard())
235      {
236          return NULL;
237      }
238      //当前节点指向第一个结点
239      if (cardList != NULL) node = cardList->next;
240
241      while (node != NULL)
242      {
243          //在链表中查找是否有对应的卡信息，找到后更新信息
244          if ((strcmp(node->data.aName, pName) == 0) && (strcmp(node->data.aPwd, pPwd) == 0))
245          {
246              //更新链表中的卡信息
247              node->data.nStatus = 1; //状态为正在使用
248              node->data.nUseCount++; //使用次数加1
249              node->data.tLastTime = time(NULL); //最后使用时间为当前时间
250          }
251      }
252  }

```

```

251 //更新文件中的卡信息
252 if (TRUE == updateCard(&node->data, CARDPATH, nIndex))
253 {
254     return &node->data;
255 }
256 else
257 {
258     return NULL;
259 }
260 }
261 node = node->next;
262 nIndex++;
263 }
264 if (node == NULL) printf("----***---无该卡信息! 请重新输入! ----***---\n");
265 return NULL;
266 }

```

在 card_file.c 中定义 updateCard 函数(同时在对头文件中添加函数声明),代码如下:

file.h	card_file.c	card_service.h	card_service.c	menu.h
accountManagement (全局范围)				

```

153 //函数名 updateCard
154 //功能 更新卡信息文件中对应的一条卡信息
155 //参数 pCard:更新后的卡内容 pPath:卡信息文件的路径
156 // nIndex:需要更新的卡信息在文件中的卡序号
157 //返回值 TRUE 更新成功, FALSE 更新失败
158 int updateCard(const Card* pCard, const char* pPath, int nIndex)
159 {
160     FILE* fp = NULL; // 文件指针
161     char aBuf[CARDCHARNUM] = { 0 };
162     char aStart[30]; //存放转换后的时间字符串
163     char aEnd[30]; //存放转换后的时间字符串
164     char aLast[30]; //存放转换后的时间字符串
165     int nLine = 0; // 文件中当前卡序号(行)
166     long lPosition = 0; // 文件位置标记
167     int isSaveScs; //存储到文件成功否
168
169     //将time_t类型时间转换为字符串,字符串格式为"年-月-日 时:分"
170     timeToString(pCard->tStart, aStart);
171     timeToString(pCard->tEnd, aEnd);
172     timeToString(pCard->tLastTime, aLast);
173
174     // 以读写模式打开文件,如果失败,返回FALSE
175     if ((fp = fopen(pPath, "r+")) == NULL)
176     {
177         printf("----***---更新卡信息打开文件失败! ----***---\n");
178         return FALSE;
179     }
180

```

```

181 // 遍历文件，获取卡在文件中位置
182 //注意nIndex是doLogon函数中获得的节点在链表（文件）中序号，从0开始编号的
183 while ((!feof(fp)) && (nLine < nIndex))
184 {
185     memset(aBuf, 0, CARDCHARNUM); //清空字符数组
186     // 逐行读取文件内容
187     if (fgets(aBuf, CARDCHARNUM, fp) != NULL)
188     {
189         // 获取文件标识位置，循环的最后一次是找到的位置，在要更新的卡信息开始位置
190         lPosition = ftell(fp);
191         nLine++;
192     }
193 }
194
195 // 移到文件标识位置
196 fseek(fp, lPosition, 0);
197 //按指定格式将信息更新到文件
198 isSaveScs = fprintf(fp, "%s###s###d###s###s###.lf###s###d###.lf###d\n",
199                     pCard->aName, pCard->aPwd, pCard->nStatus, aStart, aEnd,
200                     pCard->fTotalUse, aLast, pCard->nUseCount, pCard->fBalance, pCard->nDel);
201
202 if (isSaveScs > 0) printf("----*---卡信息更新到文件成功! ----*---\n\n");
203
204 // 关闭文件
205 fclose(fp);
206 return TRUE;
207 }

```

程序中使用了读写模式打开文件，文件既可以读，也可以写。此处，先读文件，找到要更新的卡信息处，再写文件，将新的卡信息覆盖原来的卡信息。

输入输出文件时有一个文件位置指针（文件内部指针，指示当前要读/写文件的位置，实际上不是一个指针，是相对于文件开头位置的偏移量），程序中使用了 `ftell` 和 `fseek` 两个函数，分别用来得到当前文件位置指针的位置和设置当前文件位置指针的位置。

`ftell` 函数，返回一个 `long` 类型的值(整数)，表示位置指针当前位置(用相对文件开头的偏移量表示)，失败返回 `-1L`

`fseek` 函数，改变文件位置指针到指定的位置，位置由后面的参数确定。成功，返回 `0`；失败，返回非 `0` 值

思考：

- 1) 语句 `fseek(fp, lPosition, 0);` 中各参数的含义是什么？
- 2) 程序中 `while` 循环的起什么作用？ `while` 循环结束时文件位置指针在什么位置？
- 3) 找到要修改卡信息的文件位置后，为什么还要再次设置文件位置指针？将语句 `fseek(fp, lPosition, 0);` 删除，执行多个不同卡的上机操作后，到 `card.txt` 文件中去查看卡信息的更新情况，得出自己的结论，所以该语句可以去掉吗？

3. 程序优化

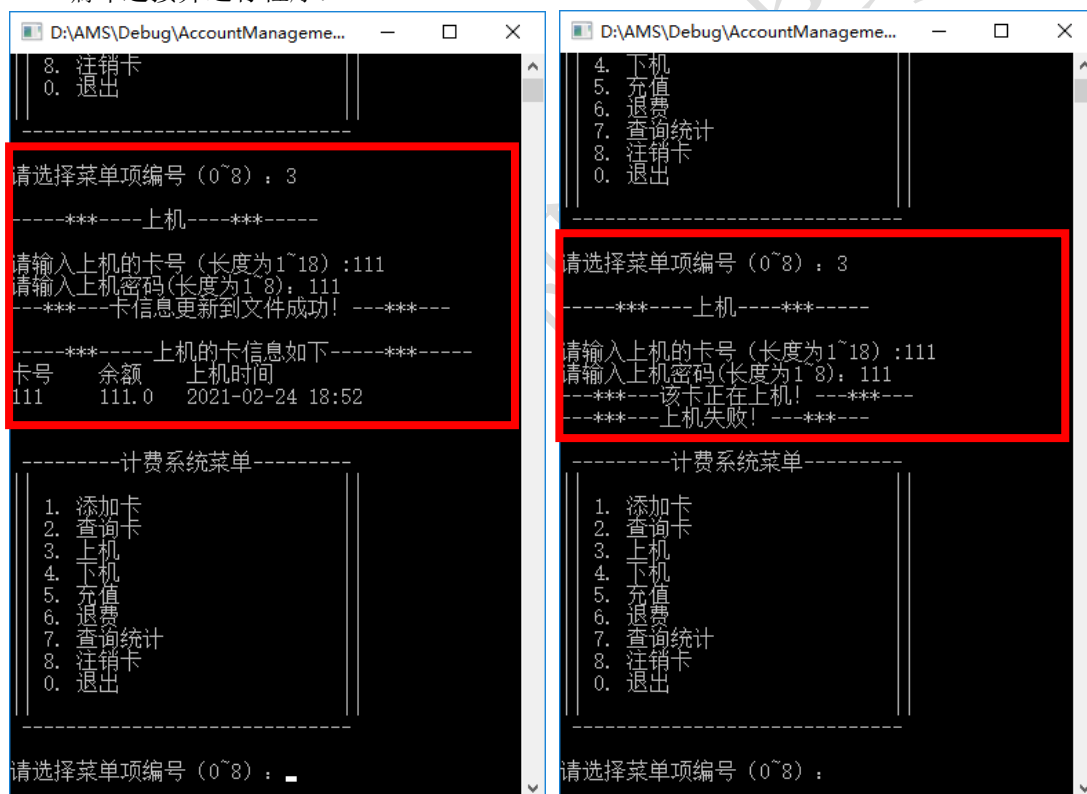
上机的条件，除了卡号和密码匹配，还有其他一些条件需要满足，修改 `card_service.c` 中的 `doLogon` 函数如下：

```

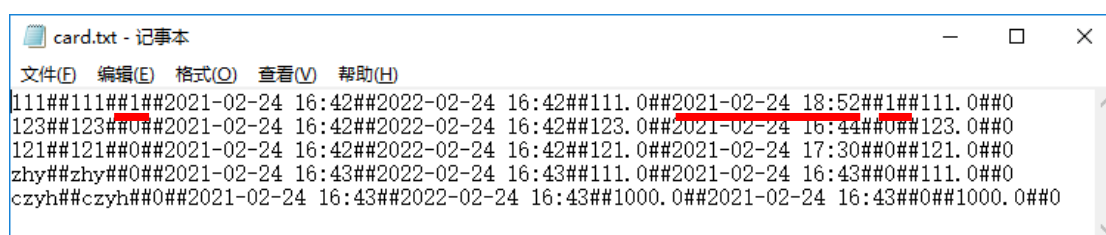
241 while (node != NULL)
242 {
243     //在链表中查找是否有对应的卡信息，找到后更新信息
244     if ((strcmp(node->data.aName, pName) == 0) && (strcmp(node->data.aPwd, pPwd) == 0))
245     {
246         //只有未上机的卡才能上机
247         if (node->data.nStatus != 0)
248         {
249             printf("-----该卡正在上机! -----\n");
250             return NULL;
251         }
252         //余额大于0的才能上机
253         if (node->data.fBalance <= 0)
254         {
255             printf("-----余额不足，请充值后再上机! -----\n");
256             return NULL;
257         }
258
259         //更新链表中的卡信息
260         node->data.nStatus = 1; //状态为正在使用
261         node->data.nUseCount++; //使用次数加1
262         node->data.tLastTime = time(NULL); //最后使用时间为当前时间
263     }
}

```

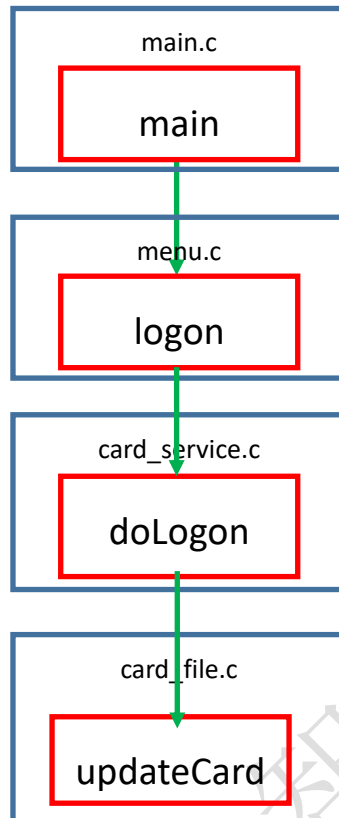
编译连接并运行程序：



查看 card.txt 文件中的变化：



本节任务的层次结构和主要调用关系



二. 添加消费记录

上机操作时，还要保存上机卡的消费信息，采用二进制文件存储。

1. 保存消费信息

在 model.h 文件中，在 #endif 之前定义保存消费信息的结构体

```

model.h  x card_file.h  card_file.c  card_service.h  ca
AccountManagement  CardNode::CardI

26  //消费信息结构体
27  typedef struct Billing
28  {
29      char aCardName[18]; //卡号
30      time_t tStart; //上机时间
31      time_t tEnd; //下机时间
32      float fAmount; //消费金额
33      int nStatus; //消费状态，0-未结算，1-已经结算
34      int nDel; //删除标识，0-未删除，1-已删除
35  }Billing;
36
37  #endif
  
```

添加 billing_file.c 和 billing_file.h 文件，在 billing_file.c 文件中定义 saveBilling 函数（同时在对应头文件添加函数声明），代码如下：

```

g_file.h  billing_file.c  model.h  card_file.h  card_file.c  card_sei
:countManagement  (全局范围)
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <stdio.h> // 包含文件处理头文件
3  #include <stdlib.h> // 包含动态内存分配头文件
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
7  // [函数名] saveBilling
8  // [功能] 保存计费信息
9  // [参数] pBilling: 计费信息结构体指针 pPath: 保存计费信息文件路径
10 // [返回值] TRUE 保存成功; FALSE 保存失败
11 int saveBilling(const Billing* pBilling, const char* pPath)
12 {
13     FILE* fp = NULL; // 文件结构体指针
14
15     // 以追加方式打开一个二进制文件
16     if ((fp = fopen(pPath, "ab")) == NULL)
17     {
18         // 如果以追加方式失败, 则以只写方式创建一个文件并打开
19         if ((fp = fopen(pPath, "wb")) == NULL)
20         {
21             printf("-----添加计费信息打开文件失败! -----\n");
22             return FALSE;
23         }
24     }
25     // 将计费信息保存到文件中
26     fwrite(pBilling, sizeof(Billing), 1, fp);
27     // 关闭文件
28     fclose(fp);
29     printf("-----计费信息成功存入文件! -----\n\n");
30     return TRUE;
31 }

```

```

billing_file.h  billing_file.c  main.c
AccountManagement  (全局范围)
1  #pragma once
2  #include "model.h"
3
4  // 函数声明
5  int saveBilling(const Billing* pBilling, const char* pPath);
6

```

二进制文件的读写使用 fread 和 fwrite 函数。原型为：

```
size_t fread(void *buffer, size_t size, size_t count, FILE *fp);
```

```
size_t fwrite(void *buffer, size_t size, size_t count, FILE *fp);
```

size_t 是 unsigned int 类型；

fread 从文件一次读取 size 大小的数据，到 buffer 指向的数据块中，读取 count 次；

fwrite 从 buffer 指向的数据块写入 size 大小的数据到文件中，写入 count 次；

这里 fwrite(pBilling, sizeof(Billing), 1, fp); 就是将内存中 pBilling 指向的计费信息结构体的内容一次性写入 fp 指向的文件中去

在 global.h 文件中，定义消费信息文件的保存路径

```
global.h  billing_file.h  billing_file.c  main.c
AccountManagement (全局范围)
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  #define BILLINGPATH "data\\billing.ams" // 计费信息保存路径
```

在 data 目录下手工新建一个文本文件并修改文件名为 billing.ams，注意要按实验三中的方法将文件扩展名显示出来后再修改，文件的扩展名修改为 ams，检查一下带扩展名的文件名是否是 billing.ams，不能是 billing.ams.txt。（提示：也可以不手工建立这个文件，在第一次添加消费信息时会自动建立这个文件，但是初次运行时可能会出来一些“不能打开文件”之类的提示信息，可忽略）

AMS > AccountManagement > data

名称	修改日期	类型	大小
billing.ams	2021/3/6 18:48	AMS 文件	0 KB
card.txt	2021/2/24 18:52	文本文档	1 KB

添加 billing_service.c 和 billing_service.h 文件，在 billing_service.c 文件中定义 addBilling 函数（同时在对头文件添加函数声明），先将相关信息写入消费信息结构体，再调用 billing_file.c 中的 saveBilling 函数，将消费信息结构体的内容写入计费信息文件保存下来，代码如下：

```
service.h  billing_service.c  main.c
AccountManagement (全局范围)
1  #define _CRT_SECURE_NO_WARNINGS
2  #include <string.h>
3  #include <time.h>
4
5  #include "model.h" // 包含数据类型定义头文件
6  #include "global.h" // 包含全局定义头文件
7  #include "billing_file.h"
8
9  //函数名 addBilling
10 //功能 上机时添加消费信息到文件
11 //参数 pName: 上机卡号; pBilling: 指向消费信息结构体
12 //返回值 TRUE: 保存成功; FALSE: 保存失败
13 int addBilling(const char *pName, Billing *pBilling)
14 {
15     //消费信息写入结构体
16     strcpy(pBilling->aCardName, pName); //上机卡号
17     pBilling->tStart = time(NULL); //上机时间
18     pBilling->tEnd = 0; //下机时间
19     pBilling->fAmount = 0; //消费金额
20     pBilling->nStatus = 0; //消费状态, 0-未结算, 1-已经结算
21     pBilling->nDel = 0; //删除标识, 0-未删除, 1-已删除
22
23     //消费信息结构体写入文件
24     if (FALSE == saveBilling(pBilling, BILLINGPATH))
25     {
26         return FALSE;
27     }
28     return TRUE;
29 }
```

要调用saveBilling函数，文件前面添加#include "billing_file.h"

在 card_service.c 中修改 doLogon 函数，在更新完卡信息后，添加消费信息，添加成功才表示上机成功。

```

service.h  card_service.c  billing_service.h  billing_service.c  main.c
accountManagement (全局范围)
225  //[[函数名] doLogon
226  //[[功能] 从文件读取卡信息到链表，在链表中查询匹配的卡信息，
227  //      更新链表和文件中对应卡信息，保存计费信息
228  //[[参数]  pName: 上机卡号; pPwd: 上机密码
229  //[[返回值] 上机成功返回TRUE，失败返回FALSE
230  int doLogon(const char* pName, const char* pPwd)
231  {
232      lpCardNode node = NULL; //当前节点
233      int nIndex = 0; //要查找节点在链表中序号，用于更新卡信息
234      Billing *pBilling = NULL; //计费信息
235
236      //从卡信息文件中读取卡信息到链表，失败返回NULL
237      if (FALSE == getCard())
238      {
239          return FALSE;
240      }
241      //当前节点指向第一个结点
242      if (cardList != NULL) node = cardList->next;
243

```

```

service.h  card_service.c  billing_service.h  billing_service.c  main.c
accountManagement (全局范围) do
244  while (node != NULL)
245  {
246      //在链表中查找是否有对应的卡信息，找到后更新信息
247      if ((strcmp(node->data.aName, pName) == 0) && (strcmp(node->data.aPwd, pPwd) == 0))
248      {
249          //只有未上机的卡才能上机
250          if (node->data.nStatus != 0)
251          {
252              printf("-----该卡正在上机! -----\n");
253              return FALSE;
254          }
255          //余额大于0的才能上机
256          if (node->data.fBalance <= 0)
257          {
258              printf("-----余额不足，请充值后再上机! -----\n");
259              return FALSE;
260          }
261
262          //更新链表中的卡信息
263          node->data.nStatus = 1; //状态为正在使用
264          node->data.nUseCount++; //使用次数加1
265          node->data.tLastTime = time(NULL); //最后使用时间为当前时间
266
267          //更新文件中的卡信息
268          if (FALSE == updateCard(&node->data, CARDPATH, nIndex))
269          {
270              //文件更新失败返回，更新成功才继续添加消费记录
271              return FALSE;

```

```

272
273 //添加消费记录到文件
274 pBilling = (Billing*)malloc(sizeof(Billing));
275 if (TRUE == addBilling(node->data.aName, pBilling))
276 {
277     free(pBilling);
278     //消费信息保存成功后，则表示上机成功
279     return TRUE;
280 }
281 else
282 {
283     return FALSE;
284 }
285 }
286 node = node->next;
287 nIndex++;
288 }
289 if (node == NULL) printf("-----无该卡信息！请重新输入！-----\n");
290 return FALSE;
291 }

```

函数返回值类型改为 int（并且修改对应头文件中该函数的声明），修改函数中的返回值为 TRUE 或 FALSE

要调用 addBilling 函数，文件前面添加 `#include "billing_service.h"`

2. 保存并显示上机信息

在 model.h 文件中，在 #endif 之前定义上机信息结构体

```

36
37 //上机信息结构体
38 typedef struct LogonInfo
39 {
40     char aCardName[18]; //上机卡号
41     time_t tLogon; //上机时间
42     float fBalance; //上机时卡余额
43 }LogonInfo;
44
45 #endif

```

在 card_service.c 中修改 doLogon 函数，新增一个参数 pInfo（修改对应头文件中该函数的声明），代码中在成功添加计费信息后，将上机信息保存到上机信息结构体 pInfo 中返回，其中卡号和密码从卡信息中获取，上机时间从消费信息中获取。

d_service.h	card_service.c	billing_service.h	billing_service.c	main.c
AccountManagement (全局范围)				
226	227	228	229	230
231	232	233	234	235
273	274	275	276	277
278	279	280	281	282
283	284	285	286	287
288	289	290	291	292
293	294			

```

//[[函数名] doLogon
//[[功能] 从文件读取卡信息到链表，在链表中查询匹配的卡信息，
// 更新链表和文件中对应卡信息，保存计费信息
//[[参数] pName:上机卡号；pPwd:上机密码；pInfo:指向上机信息结构体
//[[返回值] 上机成功返回TRUE，失败返回FALSE
int doLogon(const char* pName, const char* pPwd, LogonInfo* pInfo)
{
    lpCardNode node = NULL; //当前节点
    int nIndex = 0; //要查找节点在链表中序号，用于更新卡信息
    Billing *pBilling = NULL; //计费信息

    //添加消费记录到文件
    pBilling = (Billing*)malloc(sizeof(Billing));
    if (TRUE == addBilling(node->data.aName, pBilling))
    {
        //成功上机，上机信息保存到上机信息结构体
        strcpy(pInfo->aCardName, node->data.aName);
        pInfo->tLogon = pBilling->tStart;
        pInfo->fBalance = node->data.fBalance;

        free(pBilling);
        //消费信息保存成功后，则表示上机成功
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}
node = node->next;
nIndex++;

```

在 global.h 文件中定义几个常量

global.h	card_service.h	card_service.c	billing_service.h	billing_service.c
AccountManagement (全局范围)				
1	2	3	4	5
6	7	8	9	10

```

#pragma once

#define FALSE 0
#define TRUE 1
#define UNUSE 2 //卡不能使用
#define ENOUGHMONEY 3 //余额不足

#define CARDPATH "data\\card.txt" //卡信息保存路径
#define BILLINGPATH "data\\billing.ams" //计费信息保存路径

```

在 card_service.c 中修改 doLogon 函数中的返回值：

```

247 //在链表中查找是否有对应的卡信息，找到后更新信息
248 if ((strcmp(node->data.aName, pName) == 0) && (strcmp(node->data.aPwd, pPwd) == 0))
249 {
250     //只有未上机的卡才能上机
251     if (node->data.nStatus != 0)
252     {
253         printf("-----该卡正在上机! -----\n");
254         return UNUSE;
255     }
256     //余额大于0的才能上机
257     if (node->data.fBalance <= 0)
258     {
259         printf("-----余额不足，请充值后再上机! -----\n");
260         return ENOUGHMONEY;
261     }
262

```

在 menu.c 中修改 logon 函数，删除原来的显示上机卡信息的语句，调用 doLogon 函数后，根据返回值，提示显示上机的不同信息；若上机成功，从返回的上机信息结构体中获取上机信息，列表显示上机信息。代码修改如下：



```

menu.c  global.h  card_service.h  card_service.c  billing
AccountManagement (全局范围)
203 // [函数名] logon
204 // [功能] 输入上机卡信息，查询到，输出上机卡信息
205 // [参数] void
206 // [返回值] void
207 void logon()
208 {
209     char aName[18] = { 0 }; // 上机卡号
210     char aPwd[8] = { 0 }; // 上机卡密码
211     // Card *pCard = NULL; // 删掉
212     char aLastTime[30]; // 存放指定格式字符串的时间
213     LogonInfo *pInfo = NULL;
214     int nResult = 0;
215
216     // 提示并接收上机卡号
217     printf("请输入上机的卡号 (长度为1~18):");
218     scanf("%s", aName);

```

```

235
236 //开始上机，获取上机结果
237 // pCard = doLogon(aName, aPwd); // 删掉
238 pInfo = (LogonInfo*)malloc(sizeof(LogonInfo));
239 //根据上机结果，提示输出不同信息
240 nResult = doLogon(aName, aPwd, pInfo);
241
242 switch (nResult)
243 {
244     case 0:
245         printf("-----上机失败! -----\n"); break;
246     case 1:
247         printf("-----上机的卡信息如下-----\n\n");
248         // 输出表格的表头
249         printf("卡号\t余额\t上机时间\n");
250         //将time_t类型时间转换为字符串，字符串格式为“年-月-日 时:分”
251         timeToString(pInfo->tLogon, aLastTime);
252         //输出查到的卡信息
253         printf("%s\t%.1f\t%s\n", pInfo->aCardName, pInfo->fBalance, aLastTime);
254         printf("-----上机成功! -----\n");
255         break;
256     case 2:
257         printf("-----该卡不能使用! -----\n"); break;
258     case 3:
259         printf("-----余额不足! -----\n"); break;
260 }
261 printf("-----*****\n\n");
262 //释放上机信息
263 free(pInfo);
264 }
265

```

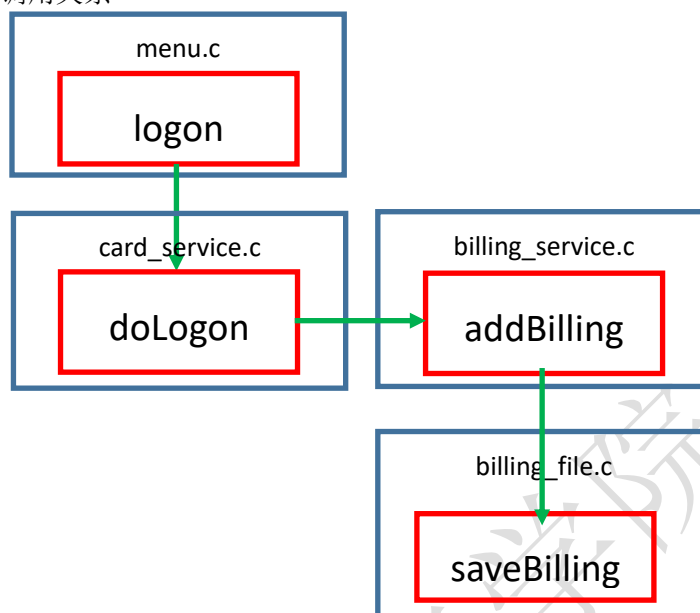
编译并运行程序

```

D:\AMS\Debug\AccountManagement.exe
0. 退出
请选择菜单项编号 (0~8): 3
-----上机-----
请输入上机的卡号 (长度为1~18): 123
请输入上机密码 (长度为1~8): 123
-----卡信息更新到文件成功! -----
-----计费信息成功存入文件!-----
-----上机的卡信息如下-----
卡号    余额    上机时间
123     123.0   2021-03-06 21:35
-----上机成功! -----
-----*****-----
-----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):

```

本节任务的层次结构和主要调用关系



三. 调整程序结构

程序中涉及到同层次的两个业务逻辑处理，卡信息管理 `card_service.c` 和消费信息管理 `billing_service.c`，处理上机操作时，`card_service.c` 要调用 `billing_service.c` 中的函数，混淆了程序的层次和功能。

新建 `service.c` 文件（以及 `service.h` 头文件），分别管理 `card_service.c` 和 `billing_service.c` 文件，让 `card_service.c` 只处理与卡信息链表相关的业务逻辑，`billing_service.c` 只处理与消费信息链表相关的业务逻辑。

1) 添加 `service.c` 文件，在其中定义 `addCardInfo` 函数（对应头文件添加函数声明）处理添加卡信息的功能。其中调用 `card_file.c` 文件的 `saveCard` 函数，将卡信息保存到文件。

```

ce.h  service.c  X
:countManagement  (全局范围)
1  #include "model.h"
2  #include "global.h"
3  #include "card_file.h"
4
5  //[函数名]addCardInfo
6  //[功能] 添加卡信息
7  //[参数] 卡信息结构体
8  //[返回值] TURE 保存成功；FALSE 保存失败
9  int addCardInfo(Card card)
10 {
11     // 将卡信息保存到文件中
12     if (TRUE == saveCard(&card, CARDPATH))
13     {
14         return TRUE;
15     }
16
17     return FALSE;
18 }
  
```

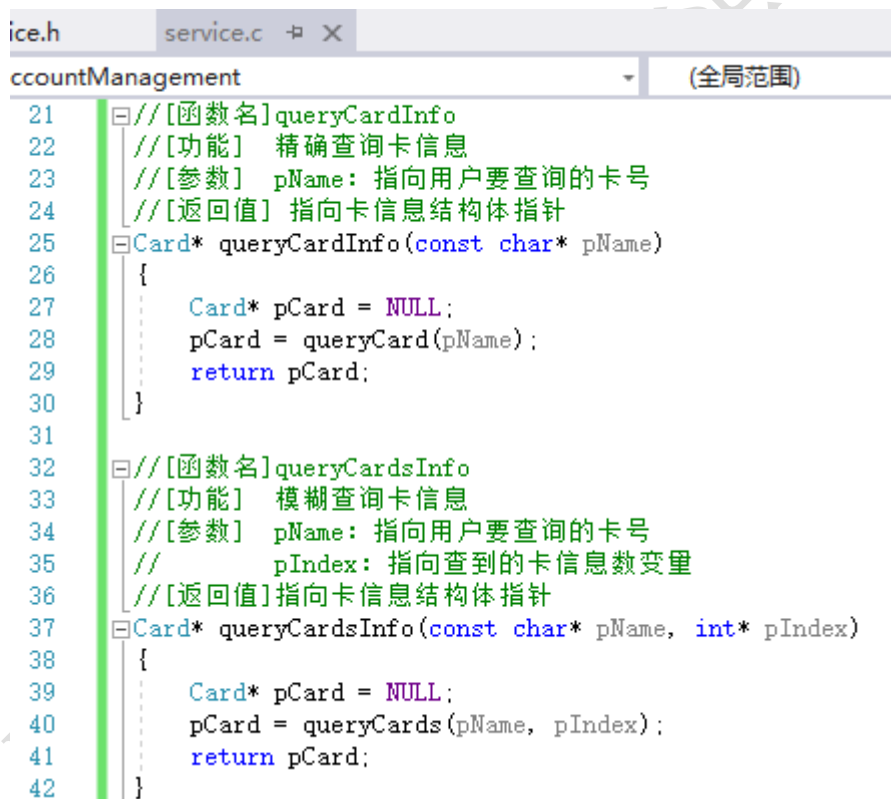
在menu.c文件中修改add函数，在最后判断保存卡信息是否成功时，原来调用addCard的位置，改为调用service.c文件中的AddCardInfo函数（文件前面添加#include "service.h"）

```

96     printf("\n-----*****-----添加的卡信息如下-----*****\n");
97     printf("\n卡号\t\t密码\t\t状态\t\t开卡金额\n");
98     printf("%s\t\t%s\t\t%d\t\t%.1f\n\n", card.aName, card.aPwd, card.nStatus, card.fBalance);
99
100    //卡信息添加到文件中
101    //if (FALSE == addCard(card))  修改为下面语句
102    if (FALSE == addCardInfo(card))
103    {
104        printf("-----*****-----添加卡信息到文件失败! -----*****\n");
105    }
106    else
107    {
108        printf("-----*****-----添加卡信息到文件成功! -----*****\n");
109    }
110 }

```

2)在service.c文件中定义queryCardInfo和queryCardsInfo函数(对应头文件添加函数声明)处理查询卡信息的功能。分别调用card_service.c文件中queryCard和queryCards函数，查询卡信息（文件前面添加#include "card_service.h"）



```

ice.h  service.c  X
ccountManagement  (全局范围)
21  //【函数名】queryCardInfo
22  //【功能】 精确查询卡信息
23  //【参数】 pName: 指向用户要查询的卡号
24  //【返回值】 指向卡信息结构体指针
25  Card* queryCardInfo(const char* pName)
26  {
27      Card* pCard = NULL;
28      pCard = queryCard(pName);
29      return pCard;
30  }
31
32  //【函数名】queryCardsInfo
33  //【功能】 模糊查询卡信息
34  //【参数】 pName: 指向用户要查询的卡号
35  //      pIndex: 指向查到的卡信息数变量
36  //【返回值】 指向卡信息结构体指针
37  Card* queryCardsInfo(const char* pName, int* pIndex)
38  {
39      Card* pCard = NULL;
40      pCard = queryCards(pName, pIndex);
41      return pCard;
42  }

```

在 menu.c 文件中修改 query 函数，在原来调用 queryCard 和 queryCards 函数处，分别改为调用 queryCardInfo 和 queryCardsInfo 函数。


```

147     printf("1.精确查询,2模糊查询(输入1或2):");
148     scanf("%d", &icha);
149
150     if (icha == 1) //选择精确查询
151     {
152         pCard = queryCardInfo(name);
153     }
154     else //默认其他选择模糊查询
155     {
156         pCard = queryCardsInfo(name, &nIndex);
157     }
158

```

在 menu.c 文件中修改 add 函数, 在原来调用 queryCard 函数处, 改为调用 queryCardInfo 函数。

```

58     // 判断输入的卡号是否已存在
59     if (queryCardInfo(name) != NULL)
60     {
61         printf("输入的卡号已存在! 请重新输入! \n");
62         return;
63     }

```

3) 在 service.c 文件中定义 doLogon 函数 (对应头文件添加函数声明) 实现上机功能。修改 card_service.c 文件中原有的 doLogon 函数, 函数名改为 checkCard (修改对应头文件中函数声明), 函数实现的功能是从文件中将卡信息读取出来, 添加到链表, 从链表中查找到符合条件的上机卡, 返回上机卡的信息及其在链表中的位置索引号。其他的功能移到新的 doLogon 函数。doLogon 函数中调用 checkCard 函数, 找到上机卡后, 更新卡信息, 调用 billing_file.c 中 saveBilling 函数, 添加消费记录。(service.c 文件前添加 #include "billing_file.h")

```

main.c  card_service.c  main.c  card_service.h  service.c
AccountManagement  (全局范围)
226  //函数名 checkCard
227  //功能 从文件读取卡信息到链表, 在链表中查询卡信息, 并获取其在链表中的位置
228  //参数 pName: 上机卡号; pPwd: 上机密码; pIndex: 返回卡的索引号
229  //返回值 上机卡结构体
230  Card* checkCard(const char* pName, const char* pPwd, int* pIndex)
231  {
232      lpCardNode cardNode = NULL;
233      int nIndex = 0; // 上机卡在卡信息链表中的索引号
234
235      // 如果从文件中获取卡信息失败, 则上机失败
236      if (FALSE == getCard())
237      {
238          return FALSE;
239      }
240      // 指向链表的第一个结点
241      cardNode = cardList->next;
242      // 遍历链表
243      while (cardNode != NULL)
244      {
245          // 查找上机卡, 判断卡号和密码是否正确
246          if ((strcmp(cardNode->data.aName, pName) == 0) && (strcmp(cardNode->data.aPwd, pPwd) == 0))
247          {
248              // 返回卡信息结点数据的地址
249              *nIndex = nIndex;
250              return &cardNode->data;
251          }
252          cardNode = cardNode->next;
253          nIndex++;
254      }
255      return NULL;
256  }

```

```

.c service.c
countManagement (全局范围)
47 //函数名 doLogon
48 //功能 进行上机操作
49 //参数 pName: 上机卡号; pPwd: 上机密码; pInfo: 指向上机信息结构体
50 //返回值 TRUE: 上机成功; FALSE: 上机失败
51 int doLogon(const char* pName, const char* pPwd, LogonInfo* pInfo)
52 {
53     Card* pCard = NULL;
54     int nIndex = 0; // 卡信息在链表中的索引, 用于更新卡信息
55     Billing billing; // 计费信息
56
57     // 根据卡号和密码, 从链表中获取卡信息和卡信息在链表中的索引
58     pCard = checkCard(pName, pPwd, &nIndex);
59
60     // 如果卡信息为空, 表示没有该卡信息, 上机失败
61     if (pCard == NULL)
62     {
63         return FALSE;
64     }
65
66     // 如果卡状态不为0, 表示该卡不能上机
67     if (pCard->nStatus != 0)
68     {
69         return UNUSE;
70     }
71
72     // 如果卡的余额为0, 不能上机
73     if (pCard->fBalance <= 0)
74     {
75         return ENOUGHMONEY;
76     }
77
78     // 如果可以上机, 更新卡信息
79     pCard->nStatus = 1; // 状态为正在使用
80     pCard->nUseCount++; // 使用次数加1
81     pCard->tLastTime = time(NULL); // 更新最后使用时间为当前时间
82
83     // 更新文件中的卡信息
84     if (FALSE == updateCard(pCard, CARDPATH, nIndex))
85     {
86         return FALSE;
87     }
88
89     // 添加消费记录
90     strcpy(billing.aCardName, pName); // 上机卡号
91     billing.tStart = time(NULL); // 上机时间
92     billing.tEnd = 0; // 下机时间
93     billing.nStatus = 0; // 消费状态
94     billing.fAmount = 0; // 消费金额
95     billing.nDel = 0; // 删除标识
96
97     // 先将计费信息保存到文件中
98     if (TRUE == saveBilling(&billing, BILLINGPATH))
99     {
100         // 组装上机信息
101         strcpy(pInfo->aCardName, pName);
102         pInfo->fBalance = pCard->fBalance;
103         pInfo->tLogon = billing.tStart;
104         return TRUE;
105     }
106     return FALSE;
107 }

```

```

service.h  service.c  X
ccountManagement (全局范围)
1  #define _CRT_SECURE_NO_WARNINGS
2  #include<string.h>
3  #include "model.h"
4  #include "global.h"
5  #include "card_file.h"
6  #include "card_service.h"
7  #include "billing_file.h"
8

```

在 menu.c 文件 logon 函数中调用 service.c 文件中的 doLogon 函数（代码不用修改）

4) 在 service.c 文件中定义 releaseList 函数（对应头文件添加函数声明）处理退出应用程序时释放链表内存，releaseList 函数调用 card_service.cpp 文件中 releaseCardList 函数

```

ce.h  menu.c  service.c  X
ccountManagement (全局范围)
108
109  // [函数名] releaseList
110  // [功能] 退出应用程序时，释放链表内存
111  // [参数] void
112  // [返回值] void
113  void releaseList()
114  {
115      releaseCardList(); // 释放卡信息链表内存
116
117  }

```

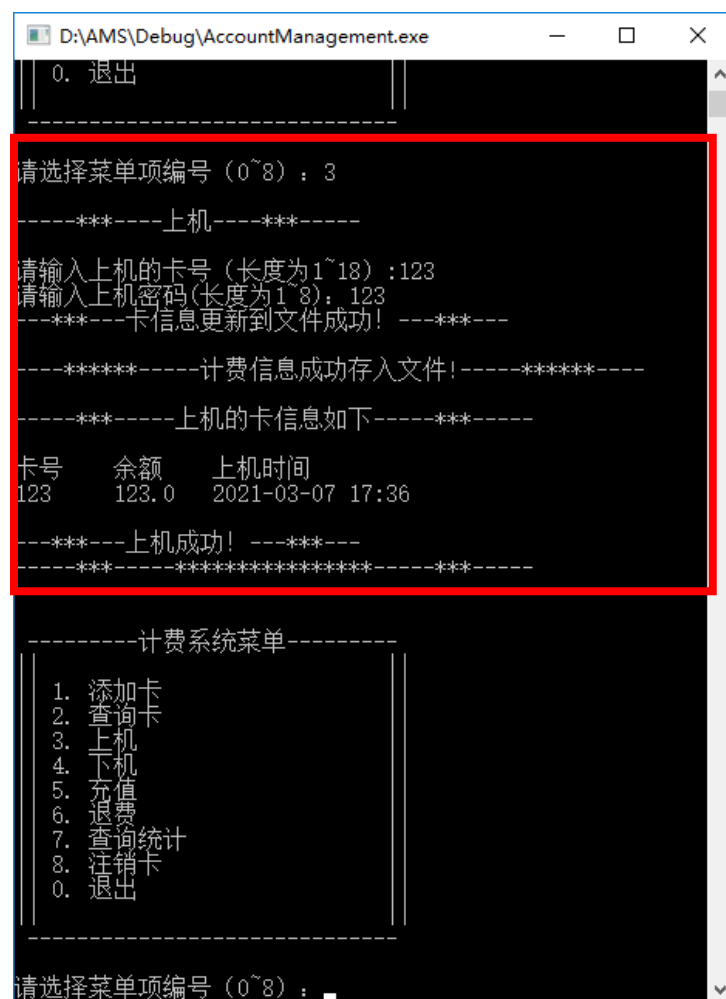
在 menu.c 文件中修改 exitApp 函数，调用 service.c 文件中 releaseList 函数

```

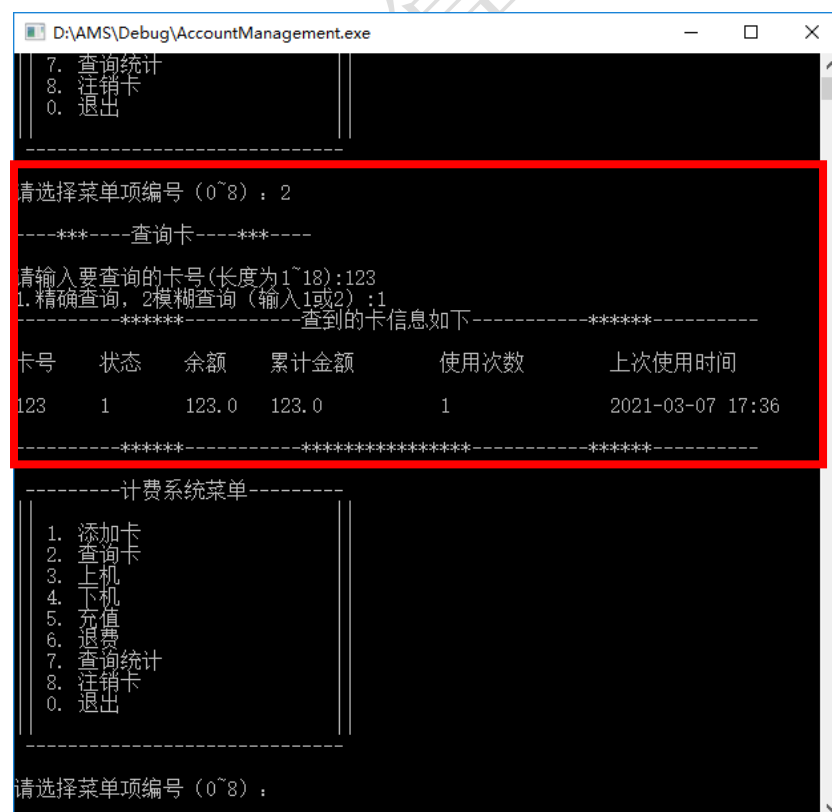
ice.h  menu.c  X  service.c
ccountManagement
196  // [函数名] exitApp
197  // [功能] 退出应用程序
198  // [参数] void
199  // [返回值] void
200  void exitApp()
201  {
202  // releaseCardList(); 删掉
203      releaseList();
204  }

```

3. 编译并运行程序



查询可以看到上机卡的状态已经改变



打开 card.txt 文件,可以看到文件中的状态也已更改

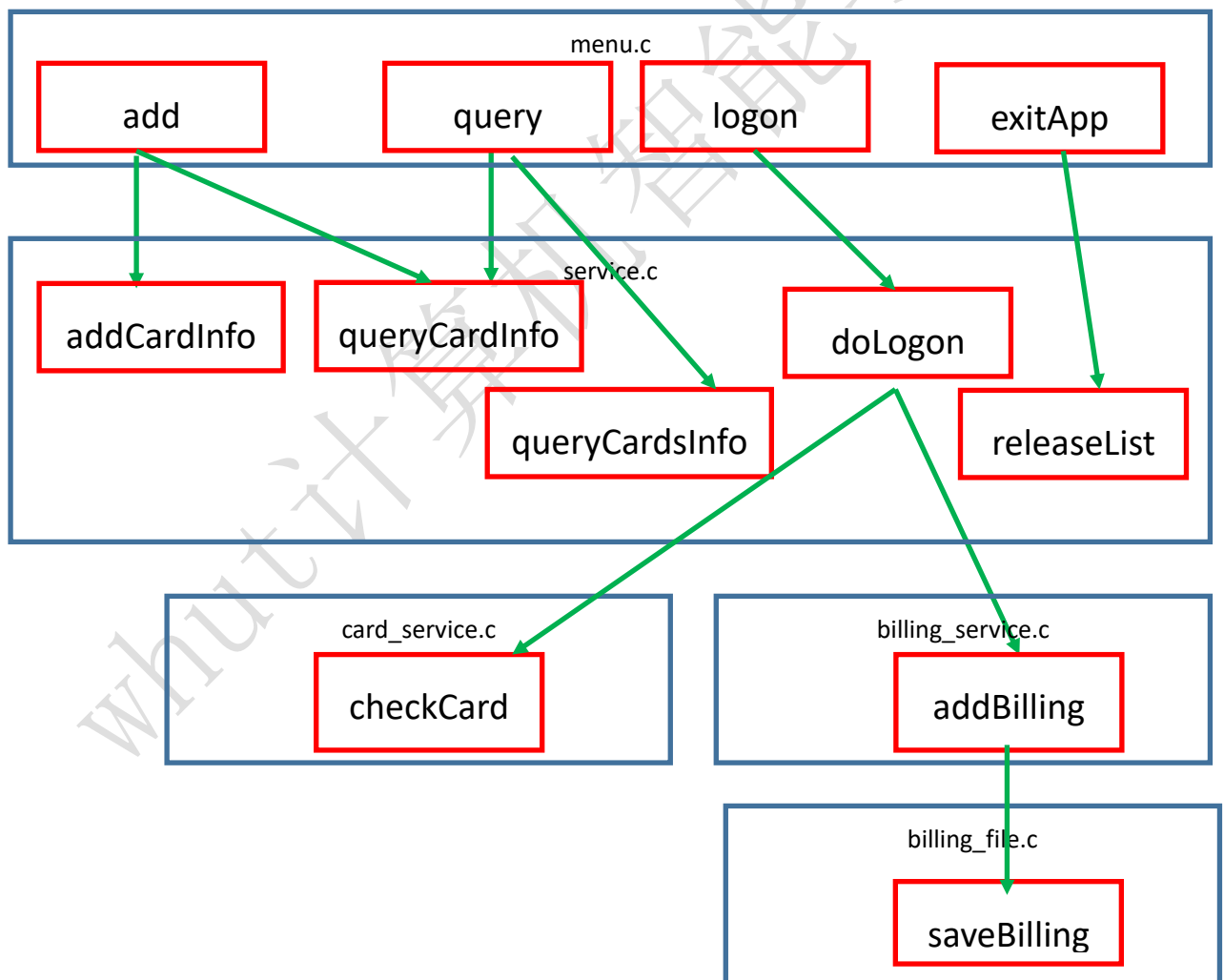
```

card.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
111##111##0##2021-02-24 16:42##2022-02-24 16:42##111.0##2021-02-24 18:52##0##111.0##0
123##123##1##2021-02-24 16:42##2022-02-24 16:42##123.0##2021-03-07 17:36##1##123.0##0
121##121##1##2021-02-24 16:42##2022-02-24 16:42##121.0##2021-03-07 17:36##1##121.0##0
zhy##zhy##0##2021-02-24 16:43##2022-02-24 16:43##111.0##2021-02-24 16:43##0##111.0##0
czyh##czyh##0##2021-02-24 16:43##2022-02-24 16:43##1000.0##2021-02-24 16:43##0##1000.0##0

```

(提示: 调试程序时多看看卡信息文件中的内容是否更改, 更新是否正确, 当计费信息文件读取有问题时, 可以删除该文件, 重新新建一个文件, 再运行程序添加卡信息和计费信息)

本节任务的层次结构和主要调用关系



四. 已完成任务的层次结构和主要调用关系

