

## 一. 定义卡信息结构体类型

## 1. 添加 model.h 头文件

## 2. 在 model.h 文件中定义卡信息结构体类型，代码如下：

```

model.h x
(全局范围)
1 #include<time.h>
2
3 struct Card
4 {
5     char aName[18];    //卡号
6     char aPwd[8];      //密码
7     int nStatus;       //卡状态 (0-未上机；1-正在上机；2-已注销；3-失效)
8     time_t tStart;     //开卡时间
9     time_t tEnd;       //卡的截止时间
10    float fTotalUse;    //累计金额
11    time_t tLastTime;   //最后使用时间
12    int nUseCount;      //使用次数
13    float fBalance;     //余额
14    int nDel;           //删除标识 (0-未删除；1-删除)
15 };

```

其中用到 `time_t` 数据类型，它的定义在 `time.h` 头文件中，所以前面要包含这个头文件

## 3. 使用 typedef 指定简化的类型名

```

model.h x
(全局范围)
1 #include<time.h>
2
3 typedef struct Card
4 {
5     char aName[18];    //卡号
6     char aPwd[8];      //密码
7     int nStatus;       //卡状态 (0-未上机；1-正在上机；2-已注销；3-失效)
8     time_t tStart;     //开卡时间
9     time_t tEnd;       //卡的截止时间
10    float fTotalUse;    //累计金额
11    time_t tLastTime;   //最后使用时间
12    int nUseCount;      //使用次数
13    float fBalance;     //余额
14    int nDel;           //删除标识 (0-未删除；1-删除)
15 }Card;

```

思考：

- 1) 为什么要在新建的 `model.h` 头文件中定义卡信息结构体类型？
- 2) `time_t` 实际上是什么类型？（查看 `time.h` 头文件中的定义）
- 3) 使用 `typedef` 的作用是什么？
- 4) `typedef struct Card` 中的 `Card` 和 最后面的 `Card` 有什么不同？

二. `time.h` 头文件中与时间相关数据的处理1. `time_t` 数据类型

time.h 中对 time\_t 的定义是 `typedef long time_t; /* 时间值 */`

## 2. time 函数，获取当前日历时间

函数原型为：`time_t time(time_t * timer);`

可以从参数 timer 返回现在的日历时间，也可以通过返回值返回现在的日历时间，即从 1970 年 1 月 1 日 0 时 0 分 0 秒到程序运行到此时的秒数数值

例：`time_t lt;`  
`lt=time(NULL); // 或 time(&lt);`

lt 中是返回的秒数数值

## 3. 时间结构体类型 tm，分解时间

time.h 中对 tm 的定义如下：

```
struct tm {
    int  tm_sec; /* 秒 - 取值区间为[0,59] */
    int  tm_min; /* 分 取值区间为[0,59] */
    int  tm_hour; /* 时 取值区间为[0,23] */
    int  tm_mday; /* 一个月中的日期 取值区间为[1,31] */
    int  tm_mon; /* 月份（从一月开始，0 代表一月） 取值区间为[0,11] */
    int  tm_year; /* 年份，其值等于实际年份减去 1900 */
    int  tm_wday; /* 星期 - 取值区间为[0,6]，其中 0 代表星期天，1 代表星期一，类推 */
    int  tm_yday; /* 从每年的 1 月 1 日开始的天数 - 取值区间为[0,365]，其中 0 代表 1 月 1 日，1 代表 1 月 2 日，类推 */
    int  tm_isdst; /* 夏令时标识符，实行夏令时的时候，tm_isdst 为正。不实行夏令时的进候，tm_isdst 为 0；不了解情况时，tm_isdst() 为负。 */
};
```

## 4. 将日历时间转换为分解时间

函数 `gmtime` 和 `localtime` 的原型为：

`struct tm * gmtime(const time_t * timer);` //返回格林威治时间，比北京时间晚八小时  
`struct tm * localtime(const time_t * timer);` //返回本地时间，即北京时间

例：`struct tm *local;`  
`time_t t;`  
`t=time(NULL);`  
`local=localtime(&t);`

local 结构体变量中是返回的本地时间的分解形式

## 5. 将分解时间转换为日历时间

`mktime` 函数原型如下：

`time_t mktime(struct tm * timeptr);`

将以年、月、日、时、分、秒等分量保存的分解时间结构，转化为日历时间的秒数数值。

## 三. 添加卡信息到结构体变量中

1. 在 menu.cpp 文件中，定义添加卡函数 add（注意别忘了在 menu.h 中添加该函数声明）
2. 在 add 函数中定义一个 Card 卡信息结构体类型的变量，用来保存用户输入的卡信息（由

于用到 Card 结构体类型，前面要添加包含 model.h 头文件)

```

menu.cpp x model.h
(全局范围)
1 #include <iostream>
2
3 #include "model.h"
4
5 using namespace std;
6
7 // [函数名]    outputMenu
8 // [功能]      输出系统菜单
9 // [参数]      void
10 // [返回值]    void
11 void outputMenu() { ... }
12
13 // [函数名]    add
14 // [功能]      添加用户卡信息到卡结构体变量，并屏幕显示
15 // [参数]      void
16 // [返回值]    void
17 void add()
18 {
19     Card card;
20 }

```

3. 在 add 函数中提示用户输入卡号，密码，开卡金额，判断合格后保存到结构体变量中，并添加结构体变量中其他成员变量的值，最后以列表格式显示出来。代码如下：

```

menu.cpp* x model.h
(全局范围)
1 #include <string.h>
2 #include <time.h>
3
4 #include <iostream>
5 #include <iomanip>
6
7 #include "model.h"
8 #include "menu.h"
9
10 using namespace std;
11
12 // [函数名]    outputMenu
13 // [功能]      输出系统菜单
14 // [参数]      void
15 // [返回值]    void
16 void outputMenu() { ... }
17
18 // [函数名]    add
19 // [功能]      添加用户卡信息到卡结构体变量，并屏幕显示
20 // [参数]      void
21 // [返回值]    void
22 void add()
23 {
24     Card card;
25 }

```

```

menu.cpp x model.h
(全局范围)
31
32 //函数名 add
33 //功能 添加用户卡信息到卡结构体变量，并屏幕显示
34 //参数 void
35 //返回值 void
36 void add()
37 {
38     Card card;
39     char name[30]; //临时存放输入的用户名
40     char pwd[20]; //临时存放输入的密码
41     struct tm* endTime; //临时存放截止时间
42     struct tm* startTime; //临时存放开卡时间
43
44     cout<<"请输入卡号（长度为1~18）:";
45     cin>> name;
46     cin.clear();
47     cin.sync();
48     //判断输入的卡号长度是否符合要求
49     if(getSize(name) >= 18)
50     {
51         cout<<"输入的卡号长度超过最大值！"<<endl;
52         return;
53     }
54     //将输入的卡号保存到卡结构体变量中
55     strcpy(card.aName, name);
56
57     cout<<"请输入密码（长度为1~8）:";
58     cin>> pwd;
59     cin.clear();
60     cin.sync();
61     //判断输入的密码是否符合要求
62     if(getSize(pwd) >= 8)
63     {
64         cout<<"输入的密码长度超过最大值！"<<endl;
65         return;
66     }
67     //将输入的密码保存到卡结构体中
68     strcpy(card.aPwd, pwd);
69
70     cout<<"请输入开卡金额(RMB): ";
71     cin>> card.fBalance;
72     cin.clear();
73     cin.sync();
74
75     card.fTotalUse = card.fBalance; //添加卡时，累计金额等于开卡金额
76     card.nDel = 0; //删除标识
77     card.nStatus = 0; //卡状态
78     card.nUseCount = 0; //使用次数
79     //开卡时间，截止时间，最后使用时间都默认为当前时间。
80     //time(NULL) 获取当前绝对时间(日历时间), 1970-01-01 00:00:00起到现在秒数
81     card.tStart = card.tEnd = card.tLastTime = time(NULL);
82
83     //根据开卡时间，计算截止时间，每张卡的有效期为一年
84     startTime = localtime(&card.tStart); //将日历时间转换为分解时间
85     endTime = localtime(&card.tEnd);
86     endTime->tm_year = startTime->tm_year + 1;
87     card.tEnd = mktime(endTime); //将分解时间转换为日历时间
88
89     cout<<endl;
90     cout<<"-----添加的卡信息如下-----"<<endl;
91     cout<<setw(12)<<"卡号"<<setw(12)<<"密码"<<setw(12)<<"状态"<<setw(12)<<"开卡金额"<<endl;
92     cout<<setw(12)<< card.aName<<setw(12)<<card.aPwd; //一行输出书写语句过长，分行
93     cout<<setw(12)<<card.nStatus<<setw(12)<<fixed<< setprecision(2)<<card.fBalance<<endl;
94 }

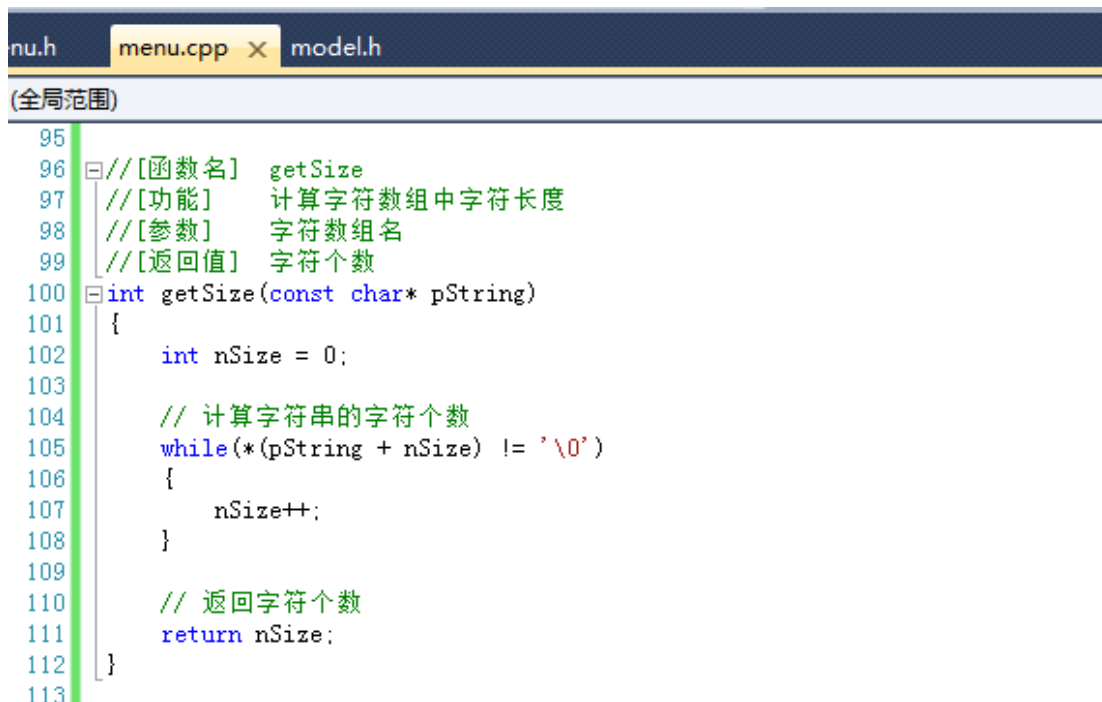
```

其中用到strcpy函数进行字符串拷贝，需添加包含头文件#include <string.h>

其中用到struct tm结构体，time函数，localtime函数，mktime函数，需添加包含头文件#include <time.h>

其中用到setw操作设置输出宽度，用fixed和setprecision设置小数格式和位数，需添加包含头文件#include <iomanip>

4. 将判断字符串长度的功能封装到函数 getSize 中（在 menu.h 中添加该函数声明），在 add 函数中直接调用，前面添加包含头文件#include“menu.h”



```

95
96 // [函数名]  getSize
97 // [功能]    计算字符数组中字符长度
98 // [参数]    字符数组名
99 // [返回值]  字符个数
100 int getSize(const char* pString)
101 {
102     int nSize = 0;
103
104     // 计算字符串的字符个数
105     while(*(pString + nSize) != '\0')
106     {
107         nSize++;
108     }
109
110     // 返回字符个数
111     return nSize;
112 }
113

```

思考：1) 字符串能直接赋值吗？

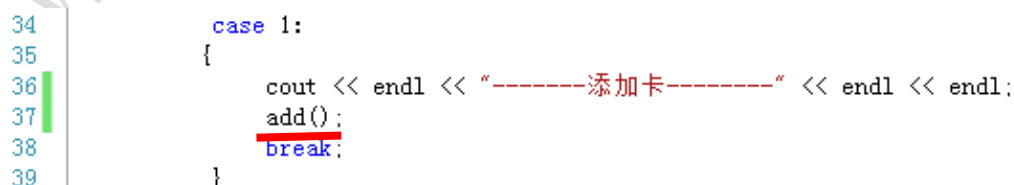
2) cout 的格式化输出有哪些？

3) 为什么 menu.cpp 前面要添加#include“menu.h”？如何修改可以不添加？

4) getSize 函数的参数类型？其中的 const 的作用？add 函数调用时传递的参数类型？是否匹配？

四调用添加卡函数

在 main.cpp 文件中的 main 函数 case 1 的分支中直接调用

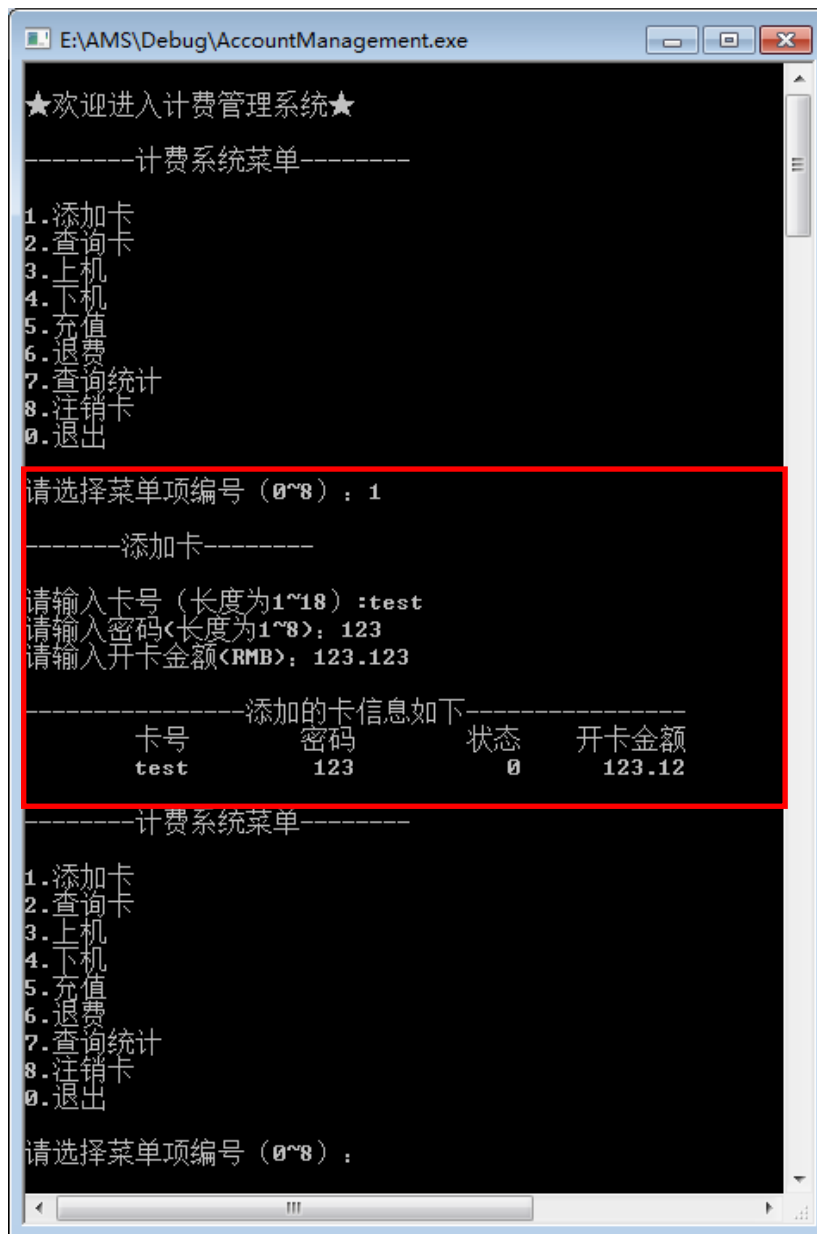


```

34         case 1:
35         {
36             cout << endl << "-----添加卡-----" << endl << endl;
37             add();
38             break;
39         }

```

五. 重新编译并运行程序



#### 六. 保存卡信息到结构体数组

1. 添加 card\_service.cpp 文件
2. 在 card\_service.cpp 中定义全局的卡信息结构体数组 aCard[50]，用来保存多条卡信息（需包含头文件 model.h）；以及全局的整型变量 nCount，用来记录数组中实际的卡信息条数。

```
card_service.cpp x menu.cpp
(全局范围)
1 #include "model.h"
2
3 Card aCard[50];    //卡结构体数组
4 int nCount=0;      //卡结构体数组中的实际卡信息数
5
```

3. 在 card\_service.cpp 中定义函数 addCard，将卡信息结构体添加到数组中。代码如下：

```

card_service.cpp x menu.cpp
(全局范围)
1  #include <string.h>
2  #include <iostream>
3  #include "model.h"
4
5  using namespace std;
6
7  Card aCard[50];    //卡结构体数组
8  int nCount=0;      //卡结构体数组中的实际卡信息数
9
10 // [函数名]    addCard
11 // [功能]      添加卡信息到结构体数组
12 // [参数]      卡信息结构体
13 // [返回值]    整数1: 添加成功; 整数0: 不能添加
14 int addCard(Card crd)
15 {
16     if (nCount<50)
17     { //数组未满, 添加一条卡信息
18         strcpy(aCard[nCount].aName, crd.aName);
19         strcpy(aCard[nCount].aPwd, crd.aPwd);
20         aCard[nCount].nStatus=crd.nStatus;
21         aCard[nCount].tStart=crd.tStart;
22         aCard[nCount].tEnd=crd.tEnd;
23         aCard[nCount].fTotalUse=crd.fTotalUse;
24         aCard[nCount].tLastTime=crd.tLastTime;
25         aCard[nCount].nUseCount=crd.nUseCount;
26         aCard[nCount].fBalance=crd.fBalance;
27         aCard[nCount].nDel=crd.nDel;
28         //计数增一
29         nCount++;
30         return 1;
31     }
32     else
33     {
34         cout<<"数组已满, 不能添加!"<<endl;
35         return 0;
36     }
37 }

```

其中用到 `strcpy` 函数进行字符串拷贝, 需添加包含头文件 `#include <string.h>`

其中用到 `cout`, 需添加包含头文件 `#include <iostream>` 以及 `using namespace std`

思考: 1) 全局变量和局部变量的区别?

2) `nCount=0` 为什么要初始化为零?

3) 虚线部分可不可以只用一条语句 `aCard[nCount]=crd;` 代替吗? 所以结构体可以直接赋值?

4. 添加 `card_service.h` 头文件, 其中添加 `addCard` 函数的声明

```
card_service.h x card_service.cpp menu.cpp
(全局范围)
1 #include "model.h"
2
3 int addCard(Card crd);
4
```

5. 在 menu.cpp 的 add 函数中调用 addCard 函数，前面要#include"card\_service.h"

```
d_service.h card_service.cpp menu.cpp x
(全局范围)
1 #include <string.h>
2 #include <time.h>
3
4 #include <iostream>
5 #include <iomanip>
6
7 #include "model.h"
8 #include "menu.h"
9 #include "card_service.h"
10
11 using namespace std;
```

Add 函数代码调整如下：

```
91 cout<<endl;
92 cout<<"-----添加的卡信息如下-----"<<endl;
93 cout<<setw(12)<<"卡号"<<setw(12)<<"密码"<<setw(12)<<"状态"<<setw(12)<<"开卡金额"<<endl;
94 cout<<setw(12)<< card.aName<<setw(12)<<card.aPwd; //一行输出书写语句过长，分行
95 cout<<setw(12)<<card.nStatus<<setw(12)<<fixed << setprecision(2)<<card.fBalance<<endl;
96
97 //卡信息添加到结构体数组中
98 if(FALSE == addCard(card))
99 {
100     cout<<"-----*****-----添加卡信息失败! -----*****-----"<<endl;
101 }
102 else
103 {
104     cout<<"-----*****-----添加卡信息成功! -----*****-----"<<endl;
105 }
106 }
```

此时编译会提示错误：

1) **Card 结构体类型重定义**，原因是包含了 2 次 model.h 中 Card 结构体的定义（一次直接在 menu.cpp，一次在 card\_service.h 中间接包含）；为避免头文件被重复包含，一般采用编译预处理命令处理头文件，目前的 3 个头文件修改处理如下（这种方式是依靠所定义的宏名不重复，来保证同一文件不被包含多次）：



```

model.h x card_service.h card_service.cpp menu.cpp
(全局范围)
1 #ifndef MODEL_H
2 #define MODEL_H
3
4 #include<time.h>
5
6 typedef struct Card
7 {
8     char aName[18];    //卡号
9     char aPwd[8];      //密码
10    int nStatus;        //卡状态 (0-未上机; 1-正在上机; 2-已注销; 3-失效)
11    time_t tStart;      //开卡时间
12    time_t tEnd;        //卡的截止时间
13    float fTotalUse;    //累计金额
14    time_t tLastTime;   //最后使用时间
15    int nUseCount;      //使用次数
16    float fBalance;     //余额
17    int nDel;           //删除标识 (0-未删除; 1-删除)
18 }Card;
19
20 #endif

menu.h x card_service.h card_service.cpp n
(全局范围)
1 #ifndef MENU_H
2 #define MENU_H
3
4 //函数声明
5 void outputMenu();
6 void add();
7 int getSize(const char* pString);
8
9 #endif
10

menu.h card_service.h x card_service.cpp
(全局范围)
1 #ifndef CARD_SERVICE_H
2 #define CARD_SERVICE_H
3
4 #include "model.h"
5
6 //函数声明
7 int addCard(Card crd);
8
9 #endif
10

```

提示：除了**#ifndef**方式，C++编译器还支持**#pragma once**方式，保证同一文件不被包含多次，只需在每个头文件开头加入该语句

方式一：

```
#ifndef MENU_H
```

```
#define MENU_H
```

// 一些声明语句

```
#endif
```

#ifndef 的方式是 C/C++语言中的宏定义，依赖于**宏名字不能冲突**。优点是**#ifndef 受语言天生的支持，不受编译器的任何限制**；缺点是如果不同头文件的宏名不小心“撞车”，可能会导致头文件明明存在，编译器硬说找不到声明的状况。

方式二：

**#pragma once****// 一些声明语句**

#pragma once 方式产生于#ifndef 之后，是编译器相关的，由编译器提供保证：同一个文件不会被包含多次。好处是不必再费劲想个宏名。缺点是如果某个头文件有多份拷贝，不能保证他们不被重复包含；不受一些较老版本的编译器支持，兼容性不够好。

或

<pre>#ifndef MENU_H #define MENU_H void outputMenu(); #endif</pre>	<pre>#pragma once void outputMenu();</pre>
--	--

2) **FALSE 未声明的标识符**，添加 global.h 头文件，其中添加宏定义，在 menu.cpp 中包含该头文件，代码如下：

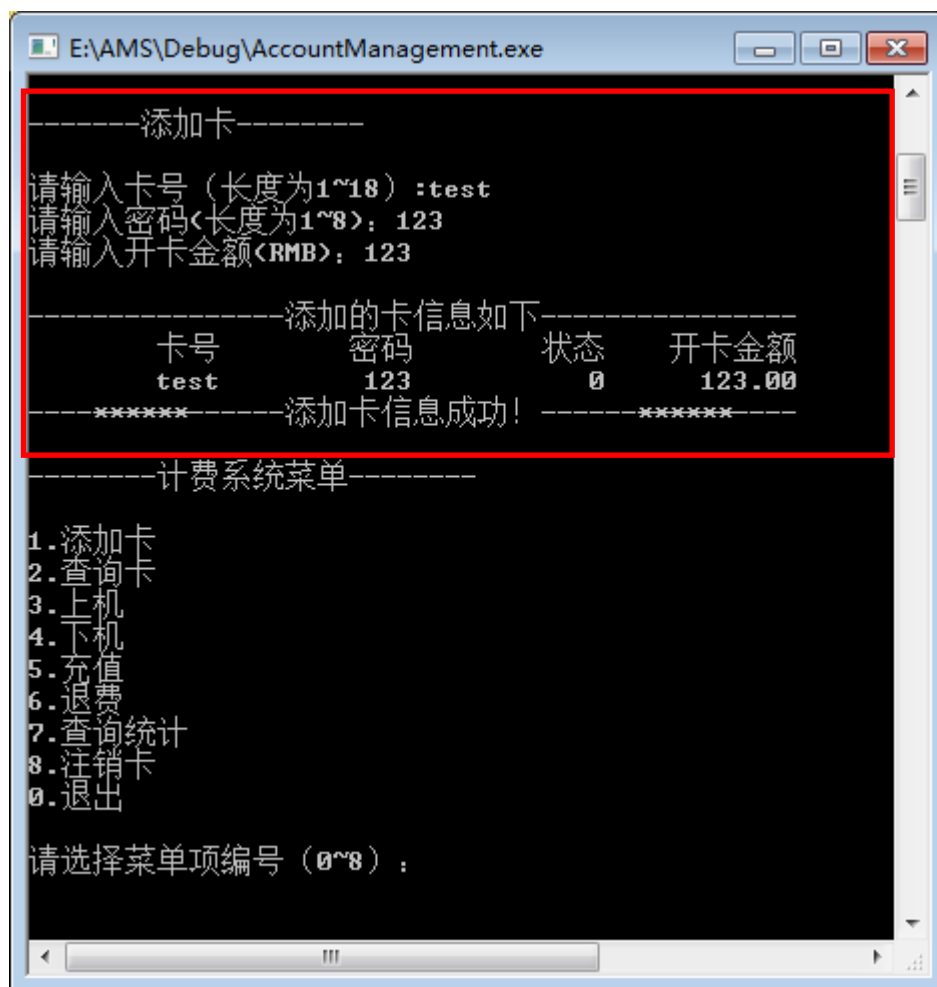
```
global.h* x card_service.cpp
(全局范围)
1 #ifndef GLOBAL_H
2 #define GLOBAL_H
3
4 #define FALSE 0
5 #define TRUE 1
6
7 #endif
```

6.修改addCard函数的返回值，1改成TRUE，0改成FALSE（前面要#include "global.h"）

思考：1) 什么情况下头文件不使用编译预处理命令，被包含多次也能编译通过？什么情况下头文件必须使用编译预处理命令？

2) 程序中为什么要用宏 FALSE 代替数字 0，TRUE 代替数字 1？

七. 重新编译并运行程序



#### 八. 定义查询卡函数并调用

1. 在 menu.cpp 中定义 query 函数，其中提示用户输入要查询的卡号，定义字符数组保存要查询的卡号（同时在 menu.h 中添加该函数声明），代码如下：

```

menu.h    card_service.cpp    menu.cpp  X
全局范围
126  // [函数名] query
127  // [功能]   根据输入的卡号，调用，查询是否有该卡，有的话，输出该卡信息
128  // [参数]   void
129  // [返回值] void
130  void query()
131  {
132      char name[18];    // 存放要查询的用户名
133
134      cout<<"请输入要查询的卡号(长度为1~18):";
135      cin>>name;
136      cin.clear();
137      cin.sync();
138
139  }
140

```

2. 在 main.cpp 的 main 函数中选择菜单项 2 时，调用 query 函数，修改 main 函数如下：

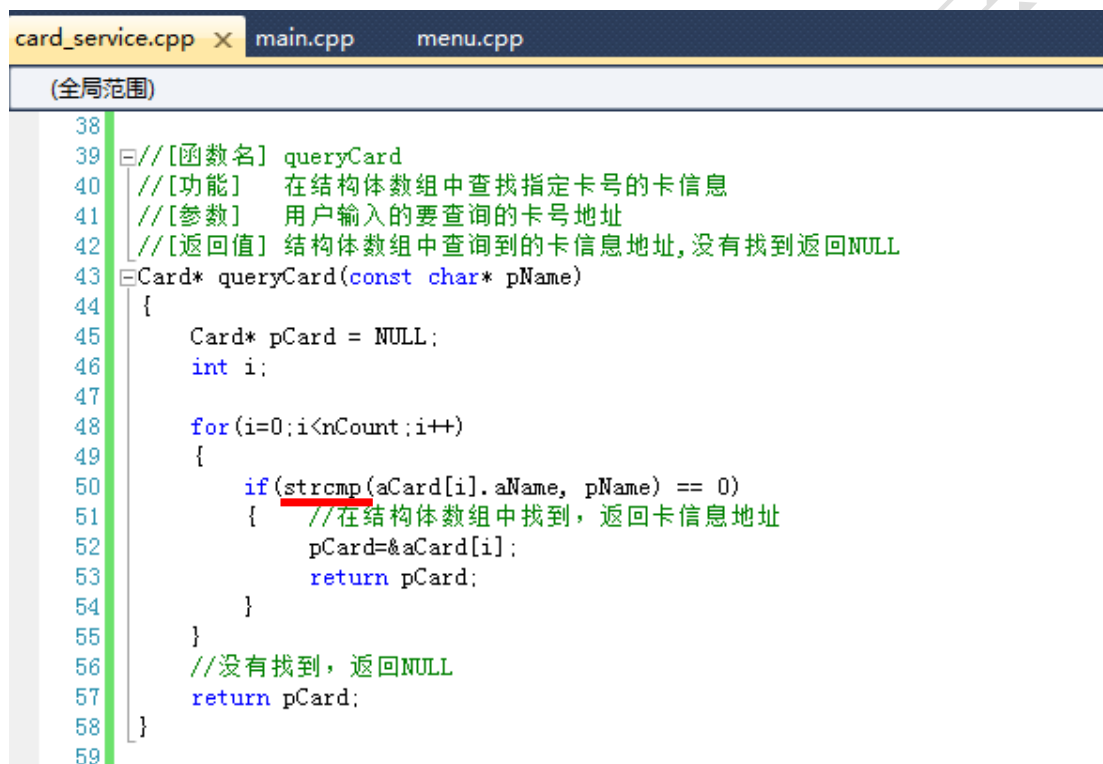
```

40         case 2:
41         {
42             cout << endl << "-----查询卡-----" << endl << endl;
43             query();
44             break;
45         }

```

九. 定义 queryCard 函数，在结构体数组中查询

1. 在 card\_service.cpp 文件中定义 queryCard 函数，参数是用户输入的要查询的卡号地址，返回值是结构体数组中查询到的卡信息地址（同时在 card\_service.h 文件中添加函数声明）；
2. queryCard 函数中，根据用户输入的卡号，在结构体数组中依次比较，第一个卡号完全相同的，即为要查找的卡信息，代码如下：



```

38
39 //函数名 queryCard
40 //功能 在结构体数组中查找指定卡号的卡信息
41 //参数 用户输入的要查询的卡号地址
42 //返回值 结构体数组中查询到的卡信息地址, 没有找到返回NULL
43 Card* queryCard(const char* pName)
44 {
45     Card* pCard = NULL;
46     int i;
47
48     for(i=0; i<nCount; i++)
49     {
50         if(strcmp(aCard[i].aName, pName) == 0)
51         { //在结构体数组中找到, 返回卡信息地址
52             pCard=&aCard[i];
53             return pCard;
54         }
55     }
56     //没有找到, 返回NULL
57     return pCard;
58 }
59

```

其中用到 strcmp 字符串比较函数，原型定义在 string.h 头文件，其参数是 2 个字符串，依次比较 2 个字符串中对应位置上字符的 ASCII 码，直到出现不同字符或某字符串结束，字符完全相同时返回零，不同时按 ASCII 码大小返回正值或负值

思考：函数调用时值传递和地址传递的不同？地址传递时形参和实参的结合方式？

十. 调用 queryCard 函数

在 menu.cpp 的 query 函数中，调用 queryCard 函数，得到查询到的卡信息，并将卡信息以列表形式显示出来（注意：结构体数组中保存的卡信息时间都是日历时间，需要转换成我们习惯的分解时间形式显示），query 函数代码如下：

```

menu.cpp x
(全局范围)
130 void query()
131 {
132     char name[18]={0};    //存放要查询的用户名
133     Card* pCard=NULL;
134     char aLastTime[30]; //存放指定格式字符串的时间
135     struct tm * timeinfo; //临时存放从time_t转换过来的tm结构时间
136
137     cout<<"请输入要查询的卡号(长度为1~18):";
138     cin>>name;
139     cin.clear();
140     cin.sync();
141
142     //从结构体数组中查找卡信息
143     pCard = queryCard(name);
144
145     // 如果pCard为NULL, 表示没有该卡的信息
146     if(pCard == NULL)
147     {
148         cout<<"-----*****-----没有该卡的信息!-----*****-----"<<endl;
149     }
150     else
151     {
152         cout<<"-----*****-----查到的卡信息如下-----*****-----"<<endl;
153         // 输出表格的表头
154         cout<<setw(10)<<"卡号"<<setw(10)<<"状态"<<setw(10)<<"余额";
155         cout<<setw(10)<<"累计金额"<<setw(10)<<"使用次数"<<setw(20)<<"上次使用时间"<<endl;
156         //将time_t类型时间转换为字符串, 字符串格式为"年-月-日 时:分"
157         timeinfo = localtime(&(pCard->tLastTime)); //time_t类型时间转换成tm结构时间
158         strftime(aLastTime, 20, "%Y-%m-%d %H:%M", timeinfo); //tm结构时间输出为指定格式字符串
159         //输出查到的卡信息
160         cout<<setw(10)<<pCard->aName<<setw(10)<<pCard->nStatus; //一行输出书写语句过长, 分行
161         cout<<setw(10)<<fixed <<setprecision(2)<<pCard->fBalance;
162         cout<<setw(10)<<fixed <<setprecision(2)<<pCard->fTotalUse;
163         cout<<setw(10)<<pCard->nUseCount<<setw(20)<<aLastTime<<endl;
164     }
165 }

```

其中用到 `strftime` 函数, 将 `tm` 类型中的时间分量按照指定格式生成字符串, 其定义在 `time.h` 头文件中, 函数原型:

`strftime (char *strDest, size_t maxsize, const char *format, const struct tm *timeptr);`

参数的含义: `strDest` 装换后的字符串; `maxsize` 字符串的最大长度;

`format` 字符串格式; `timeptr` 要转换的 `tm` 类型时间

#### 十一. 重新编译并运行程序

先执行“添加卡”, 再执行“查询卡”



## 十二.程序优化

### 1.时间的存储和显示优化

时间在卡信息结构体中是 `time_t` 类型的长整型数值，输出时以分解结构的指定字符串格式显示。新增 `tool.cpp` 文件，实现二者的装换。

1) 添加 `tool.cpp` 文件及 `tool.h` 文件（头文件中放函数声明）

2) 定义 `timeToString` 函数，将 `time_t` 格式时间，装换为“年-月-日 时：分”格式字符串，代码如下：

```

tool.h  tool.cpp x  card_service.cpp  menu.cpp
(全局范围)
1  #include <time.h>  // 包含时间类型头文件
2
3  //[[函数名] timeToString
4  //[[功能] 将time_t类型转换为字符串，字符串格式为“年-月-日 时:分”
5  //[[参数] time_t t: 需要转换的时间，char* pBuf:转换之后的字符串
6  //[[返回值] void
7  void timeToString(time_t t, char* pBuf)
8  {
9      struct tm * timeinfo;
10
11      timeinfo = localtime(&t);
12      strftime(pBuf, 20, "%Y-%m-%d %H:%M", timeinfo);
13  }
14

```

3) 修改 menu.cpp 的 query 函数，调用 timeToString 函数产生时间字符串，前面要#include “tool.h”

```

menu.cpp* x
(全局范围)  query0
131 void query()
132 {
133     char name[18]={0};    //存放要查询的用户名
134     Card* pCard=NULL;
135     char aLastTime[30]; //存放指定格式字符串的时间
136     // struct tm * timeinfo; //临时存放从time_t转换过来的tm结构时间
137
138     cout<<"请输入要查询的卡号(长度为1~18):";
139     cin>>name;
140     cin.clear();
141     cin.sync();
142
143     //从结构体数组中查找卡信息
144     pCard = queryCard(name);
145
146     // 如果pCard为NULL，表示没有该卡的信息
147     if(pCard == NULL)
148     {
149         cout<<"-----没有该卡的信息!-----"<<endl;
150     }
151     else
152     {
153         cout<<"-----查到的卡信息如下-----"<<endl;
154         // 输出表格的表头
155         cout<<setw(10)<<"卡号"<<setw(10)<<"状态"<<setw(10)<<"余额";
156         cout<<setw(10)<<"累计金额"<<setw(10)<<"使用次数"<<setw(20)<<"上次使用时间"<<endl;
157
158         //将time_t类型时间转换为字符串，字符串格式为“年-月-日 时:分”
159         timeinfo = localtime(&(pCard->tLastTime)); //time_t类型时间转换成tm结构时间
160         strftime(aLastTime, 20, "%Y-%m-%d %H:%M", timeinfo); //tm结构时间输出为指定格式字符串
161         timeToString(pCard->tLastTime, aLastTime);
162
163         //输出查到的卡信息
164         cout<<setw(10)<<pCard->aName<<setw(10)<<pCard->nStatus; //一行输出书写语句过长，分行
165         cout<<setw(10)<<fixed <<setprecision(2)<<pCard->fBalance;
166         cout<<setw(10)<<fixed <<setprecision(2)<<pCard->fTotalUse;
167         cout<<setw(10)<<pCard->nUseCount<<setw(20)<<aLastTime<<endl;
168     }
169 }

```

3) 定义stringToTime函数，将“年-月-日 时:分”格式字符串转换为time\_t格式时间，代码如下:

```

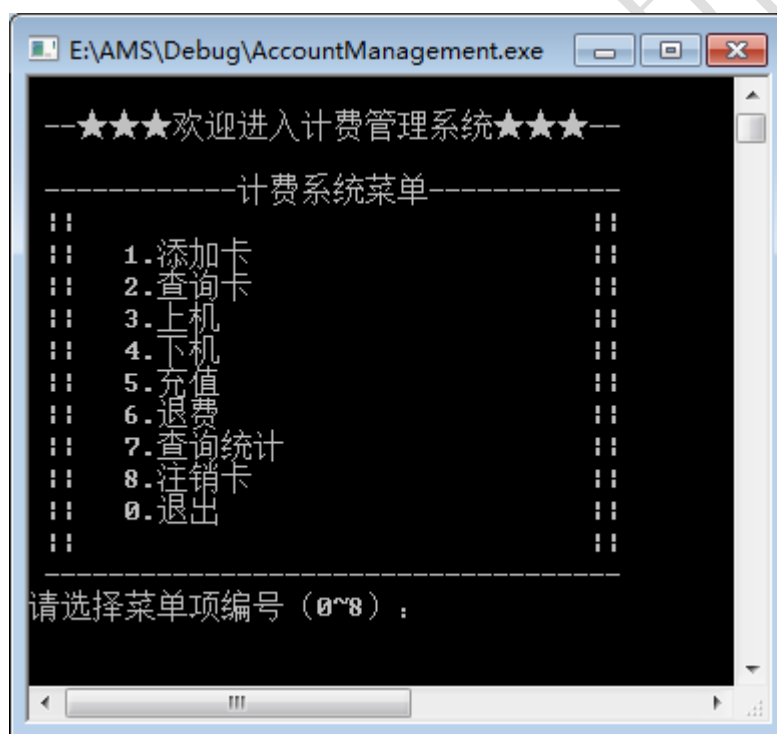
16 //[[函数名] stringToTime
17 //[[功能] 将格式为“年-月-日 时:分”的字符串转换为time_t类型时间
18 //[[参数] char* pTime: “年-月-日 时:分”格式的字符串
19 //[[返回值] time_t: 转换后的时间类型, 从1970年到该时间的秒数
20 time_t stringToTime(char* pTime)
21 {
22     struct tm tml;
23     time_t time1;
24
25     sscanf(pTime, "%d-%d-%d %d:%d", &tml.tm_year, &tml.tm_mon, &tml.tm_mday, &tml.tm_hour, &tml.tm_min);
26
27     tml.tm_year -= 1900; // 年份为从1900年开始
28     tml.tm_mon -= 1; // 月份为0~11
29     tml.tm_sec = 0;
30     tml.tm_isdst = -1;
31
32     time1 = mktime(&tml);
33
34     return time1;
35 }

```

其中使用sscanf函数（前面需#include <stdio.h>），从一个字符串中读进与指定格式相符的数据。函数原型: int sscanf( string str, string fmt, mixed var1, mixed var2 ... );

这里将字符串 pTime中对应格式“%d-%d-%d %d: %d”的5个分量分别读入tml结构体的tm\_year, tm\_mon, tm\_mday, tm\_hour, tm\_min这5个成员变量中去；最后使用mktime函数将分解时间转换为日历时间

2. 输出界面美化，字符界面的字符都堆砌在一起，不利于阅读，输出时可适当增加一些符号，空格及分隔线，例如主菜单：

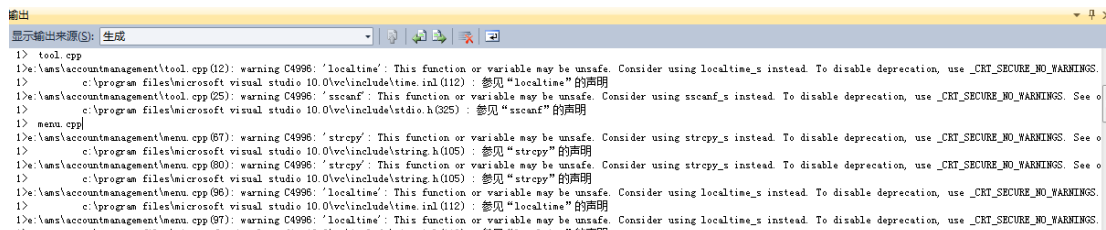


3. 为避免添加的卡号重复，在添加卡时，输入卡号后，先到结构体数组中查重，没有重复卡号才允许添加，否则重新输入卡号，修改 menu.cpp 中的 add 函数如下：

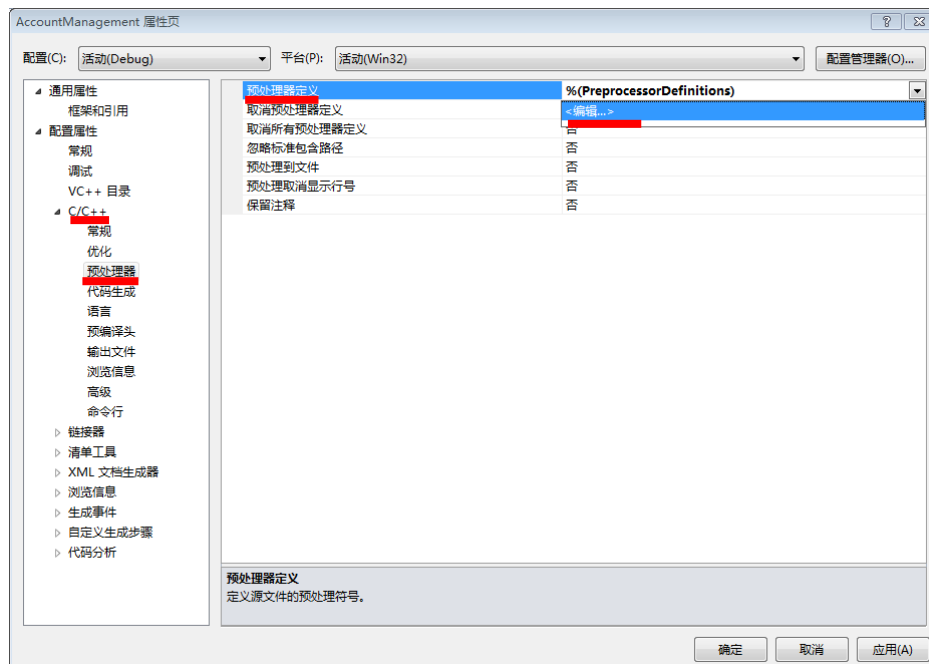




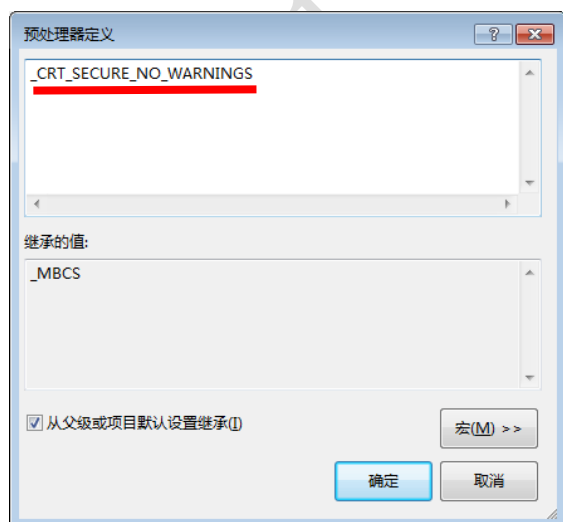
4. VS 中调用 `strcpy` 等 C 函数时输出窗口会提示 `_CRT_SECURE_NO_WARNINGS` 警告，原因是这些函数不安全，可能会造成内存泄露等。



关闭警告的方法：1) VS 中选“项目-->属性”打开对话框



2) 选“配置属性-->C/C++ --> 预处理器--> 预处理器定义--> 编辑” 打开对话框中添加 `_CRT_SECURE_NO_WARNINGS` 这个预定义，确定后即可



(或者手工在文件开头添加宏定义 `#define _CRT_SECURE_NO_WARNINGS`)

十三.总结

本次任务的层次结构和主要调用关系

