

# 中山大学实验报告

实验人：谢俊杰                  学号：18340181                  日期：2019/5/31

院（系）：数据科学与计算机学院                  专业（班级）：计科7班

实验题目：校园电子卡管理系统

## 一. 实验目的

本实验面向 C++语言的初学者。

主要让实验者熟悉面向对象的编程思想以及类的使用。

## 二. 实验环境

本实验基于Visual Studio平台开发，参考主流的编码规范，如Google C++Style Guide（中文版）

### 2.1 编程语言和开发工具

编程语言：ANSI C++

开发工具：Visual Studio Code

### 2.2 编码规范

遵循良好的程序设计风格来设计和编写程序。基本编码规范：

1. 标识符的命名要到达顾名思义的程度；
2. 关键代码提供清晰、准确的注释；
3. 程序版面要求：
  - a) 不同功能块用空行分隔；
  - b) 一般一个语句一行；
  - c) 语句缩进整齐、层次分明。

## 三. 实验内容

使用面向对象的编程思想编写一个校园电子卡管理系统，并用类的继承实现 3 种不同类别的卡：校园卡、储蓄卡、绑定卡，实现基本功能支付、查询信息和流水账、转入、转出、用文件记录数据等，扩展合理的还款、查询绑定的储蓄卡、可再次绑定储蓄卡、转帐时可根据用户需要选择校园卡或绑定卡等功能以使系统更为完善。

## 四. 分析与设计

在设计该系统时，我先写出 `card` 基类的部分成员函数和数据，然后是 3 个派生类，而且我是采用先登录后选择操作的形式，所以一开始我便将成员函数尽可能简化地写下，仅对数据成员进行简单余额变动等操作。不过在这要求绑定卡的校园卡和储蓄卡不能再是前面所说两类卡的其一，即添加为校园卡的不能作绑定卡，添加为储蓄卡的不能被绑定。这也是该程序的一个缺点。

而其次，对于文件的记录和读取，由于信息的记录顺序和数目是一定的，故我将一定数量的信息按自己的顺序写进文件中，同时又可以按定下的顺序读取出来，在这个功能的思考上，我采用了 `project1` 仓库管理系统的思路：系统运行时，先读取信息文件进链表存储以方便遍历使用，系统关闭时，再将其原文件清空遍历链表写进信息文件中。如：

```
//校园卡读取信息
ifstream in1("campus_card_inf.txt", ios::binary | ios::app | ios::in
| ios::out); //防止写入时清空文件
while (!in1.eof()) {
    string s,s1;
    in1 >> s;
    long _id;
    if (s == "")
        break;
    else {
        _id = atoi(s.c_str());
        campus_list *p = new campus_list;
        p->next = head1;
        p->user.id = _id;
        in1 >> p->user.date >> p->user.name >> s1 >> p->user.balance;
        p->user.setCollege(s1);
        head1 = p;
    }
}
```

```

    }
    in1.close();

//campus_card 写入信息
ofstream out1("campus_card_inf.txt");//写入时清空，防止重复
campus_list *p = head1;
while (p != NULL) {
    out1 << left << setw(20) << p->user.id
        << left << setw(20) << p->user.date
        << left << setw(12) << p->user.name
        << left << setw(12) << p->user.getCollege()
        << left << setw(12) << p->user.balance << endl;
    p = p->next;
}
out1.close();

```

但对于流水账的记录和读取，因为每个用户的流水账记录数目是不定且无序的，所以不能采用上述方法，而采用一用户一文件的方式，以其独一无二的 id 进行文件命名，操作时记录，查询时打开读取。如：

```

//将流水账写入以学号为名的txt文件中
void campus_card::Wr_file_rec(char s[], string operation)
{
    ofstream out1(s, ios::binary | ios::app | ios::in | ios::out);
    out1 << left << setw(20) << operation
        << left << setw(20) << amount
        << left << setw(20) << get_time()
        << left << setw(20) << balance << endl;
    out1.close();
}

//读取流水账
void Re_file_rec(char s[], int a)
{
    cout << left << setw(20) << "operation"
        << left << setw(20) << "amount"
        << left << setw(20) << "time"
        << left << setw(20) << "balance";
    if (a == 1) { //campus_crad
        cout << endl;
        ifstream in1(s, ios::binary | ios::app | ios::in | ios::out);
        while (!in1.eof()) {
            string s1;
            in1 >> s1;

```

```

        if (s1 == "")
            break;
        for (int i = 0; i < 3; i++) {
            cout << left << setw(20) << s1;
            in1 >> s1;
        }
        cout << left << setw(20) << s1 << endl;
    }
    cout << endl;
    in1.close();
}

```

最后，是对 main 函数的实现。为简洁起见，main 函数仅由两个函数组成

```

int main()
{
    Re_file_inf();
    menu();
    return 0;
}

```

menu()函数是主菜单的打印与选择，但由于需要用户选择 3 类卡且 3 类卡各自功能均不相同，故又拆分为 3 个子菜单。进入子菜单前需要“登录”，即在链表中遍历选出对应的对象，并作为之后函数的参数传递。刚开始我是采用了按值传递，在测试中发现操作不能改变余额，后才采用按引用方式传递并在完成每一操作后重新赋值给链表中对应成员，改变其信息。

#### 4.1 、需求分析：

由于该电子卡管理系统中每一种卡都有支付、查询信息和流水账的功能，所以我创建一种 card 类，内含支付函数、查询函数，同时需要查询的信息中有卡的 id（学号或卡号）、建卡日期、姓名、余额和流水账中操作的金额数，故我都将上述数据作为 card 类的 public 的数据成员，同时在卡进行每种操作时我将其记录均写进 id.txt 的文件中（没有则会自动创建），所以我在 card 类中增加一个将流水账写入文件的函数，并且上述三个函数都作为虚函数，然后每种卡均继承 card 类，使得这些函数和数据成员都能继承下来。

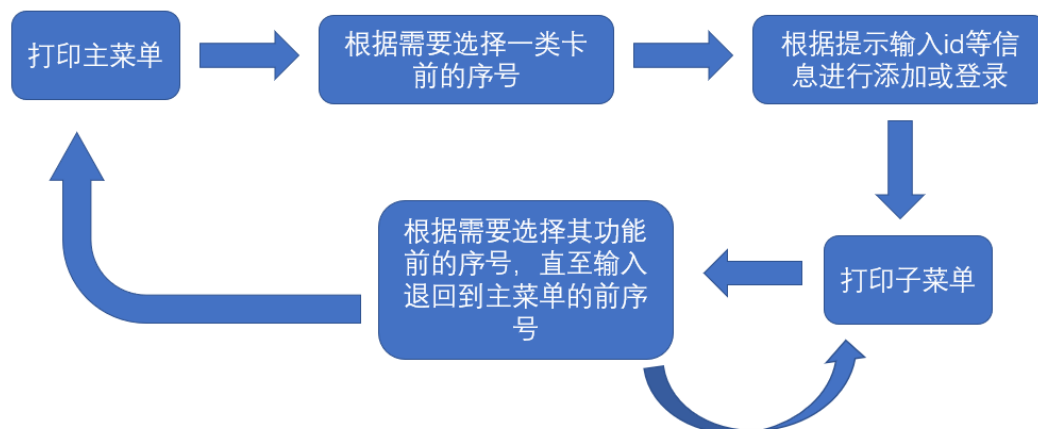
根据要求，校园卡和储蓄卡均继承 card 类，绑定卡同时继承校园卡类和储蓄卡类，而根据校园卡和储蓄卡的对比，校园卡多了学院的信息且不能透支，故在 campus\_card 类中将学院信息作为 private 的成员，而储蓄卡多了可以透支的功能，

故在 `deposit_card` 类中将透支额度作为其 `private` 的成员。而对于 `binding_card`，有着其特别的支付功能和与储蓄卡绑定的功能，但在此，我将绑定储蓄卡视为在 `private` 的成员中增加 `deposit_card` 类的数组来记录绑定的储蓄卡，在其特殊支付功能中，在校园卡余额不足时，可以自行选择其绑定的储蓄卡支付，同时可以查询已绑定的储蓄卡的信息。

在题目的最后，要求记录下流水账且可以重复查看，所以我将在每种卡的每一个操作成功之后都会将操作、金额、时间、余额（、已使用透支额度、剩余透支额度）这些数据按顺序记录进以 `id` 命名的 `txt` 文件中，需要时即可读取其文件，打印出流水账。

对于卡的信息也一样，但不同的是，我将每类卡的信息都放在每类对应的文件中，即所有校园卡的用户信息都放在 `campus_card_inf.txt` 中，在系统运行时，首先将每类卡的信息读取进 3 个链表中储存起来，方便使用其中功能，而对于新建用户时，也是存在链表中，在系统关闭时，会将链表中所有信息写入各类文件中，清空链表，释放内存。

### 系统功能图



主菜单：

```

~~~~~
~                                     ~
~   Welcome to CARD v1.0.0           ~
~   Menu:                             ~
~       1.campus_card                 ~
~       2.deposit_card                ~
~       3.binding_card                ~
~       4.Exit                        ~
~                                     ~
~~~~~

```

校园卡子菜单:

```

~~~~~
~                                     ~
~   SubMenu:                          ~
~       1.pay                         ~
~       2.inquire the information of your card ~
~       3.inquire the transactions    ~
~       4.deposit                     ~
~       5.return to the main menu     ~
~                                     ~
~~~~~

```

储蓄卡子菜单:

```

~~~~~
~                                     ~
~   SubMenu:                          ~
~       1.pay                         ~
~       2.inquire the information of your card ~
~       3.inquire the transactions    ~
~       4.deposit                     ~
~       5.transfer out                ~
~       6.repay                       ~
~       7.return to the main menu     ~
~                                     ~
~~~~~

```

绑定卡子菜单:

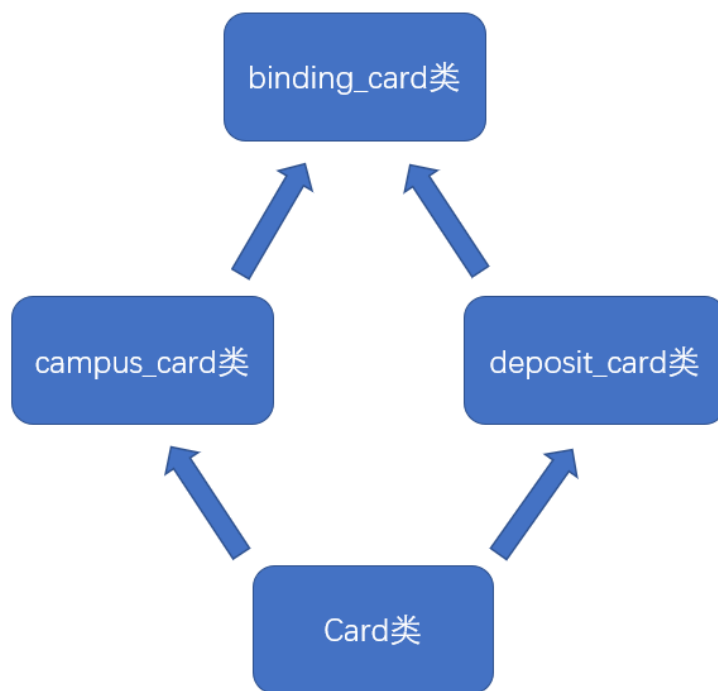
```

~~~~~
~                                     ~
~   SubMenu:                          ~
~       1.pay                         ~
~       2.inquire the information of your card ~
~       3.inquire the transactions    ~
~       4.deposit                     ~
~       5.transfer out                ~
~       6.show binding deposit cards  ~
~       7.repay                       ~
~       8.bind deposit_card           ~
~       9.return to the main menu     ~
~                                     ~
~~~~~

```

## 4.2、类结构设计

类关系图:



### 4.3、细节设计

#### 接口设计：

在接口方面，类中的成员函数如支付、转账等仅用金额作为接口，使类成员函数尽可能的简单，如：

```
void pay(double amount); //支付
void transfer_in(double amount); //只能通过储蓄卡转账存入
```

但在菜单中的函数，由于我设计的系统是先选择卡的类别再输入 id 登录的，所以这些函数的接口我是用每类卡产生的对象，按引用方式进行传递，使得在函数中改变对象的成员时作用于原对象，如

```
//campus_card 支付
void _pay1(campus_card &temp);
///campus_card 的子菜单
void submenu1(campus_card &temp);
```

#### 成员函数：

由上述分析可知, card 类的成员函数包括支付、查询、写入文件 3 个虚函数, 还有将一些数据成员初始化为 0 的缺省构造函数:

```
public:
    card(){balance = 0; amount = 0;}
    ~card(){}
    virtual void pay(double amount){} //支付函数
    virtual void inquire(int a){} //查询函数
    virtual void Wr_file_rec(char s[], string operation){} //写入流水
    账
```

而以下 3 类均是由 card 类派生, 且根据需求, 增加相应函数, 使其功能更为合理。

campus\_card 类的成员函数是:

```
public:
    campus_card();
    void pay(double amount); //支付
    void inquire(int a); //查询
    void Wr_file_rec(char s[], string operation); //写入
    void transfer_in(double amount); //只能通过储蓄卡转账存入
    void setCollege(string college); //设置学院
    string getCollege(); //获取学院
```

deposit\_card 类的成员函数是:

```
public:
    deposit_card();
    void pay(double amount); //支付
    void inquire(int a); //查询
    void Wr_file_rec(char s[], string operation); //写入
    void deposit(double amount); //存储
    bool transfer_out(double amount); //转账到另一卡
    void setUsed_q(double num); //设置已用额度
    void setSpare_q(double num); //设置剩余额度
    double getUsed_q(); //获取已用额度
    double getSpare_q(); //获取剩余额度
    void repay(); //还款
```

binding\_card 类的成员函数是:

```
public:
    binding_card();
    void pay(double amount); //特殊支付功能
    void inquire(int a); //查询
    void Wr_file_rec(char s[], string operation); //写入
    void setBinding_card(long _id); //设置绑定储蓄卡的 id
    void show_cards(); //打印出已绑定的储蓄卡
```



```
deposit_card getdeposit_card(long _id); //获取对应 id 的储蓄卡
bool isd_card(long _id); //判断是否存在该储蓄卡
int getNum(); //获取储蓄卡数目
string getId(); //获取储蓄卡 id
```

### 数据成员：

在基类 card 类中，由于各类卡都由其派生，故其数据成员均是一些较为基本的共同数据：

```
long id; //学号或卡号
string date; //建卡日期
string password; //密码
string name; //姓名
double balance; //余额
double amount; //收入 (+) 或支出 (-)
```

campus\_card 类的数据成员新增了学院：

```
private:
    string college; //学院
```

deposit\_card 类的数据成员则新增有关透支额度（我将全部额度均设为 250 元）：

```
private:
    double total_quota; //总透支额度
    double used_quota; //已使用透支额度
    double spare_quota; //剩余透支额度
```

对于 binding\_card 类，新增绑定的储蓄卡及其数目，且鉴于实际情况，绑定的储蓄卡应不会超过 20 张，数量较少，所以采用数组形式保存：

```
private:
    deposit_card binding_id[20]; //绑定的储蓄卡
    int num; //储蓄卡的数目
```

## 五、实验结果

运行该管理系统时，打印出菜单：

```

~
~ Welcome to CARD v1.0.0 ~
~ Menu: ~
~ 1.campus_card ~
~ 2.deposit_card ~
~ 3.binding_card ~
~ 4.Exit ~
~
~
Please enter what you want to do next:

```

由于校园卡只能通过储蓄卡存入，先选择 2，创建储蓄卡：

```

Please enter what you want to do next:
2
Please enter the id of your card or add your card into this system:
1621193458
There is no such id in this system. To add your id in it, please enter your name:
xjj
~
~ SubMenu: ~
~ 1.pay ~
~ 2.inquire the information of your card ~
~ 3.inquire the transactions ~
~ 4.deposit ~
~ 5.transfer out ~
~ 6.repay ~
~ 7.return to the main menu ~
~
~
Please enter what you want to do next:

```

根据提示输入 id、名字后，成功添加储蓄卡，进入储蓄卡的子菜单，选择 4 进行现金存钱：

```

Please enter what you want to do next:
4
Transfer in by 1.cash      2.deposit_card
1
Please enter the amount you will transfer in:
1000
You have deposit successfully!

```

存钱 1000 元成功后，分别选择 2 和 3 进行查询信息和流水账：

```

Please enter what you want to do next:
2
id          date          name          balance    used_quota
1621193458  2019/06/01/14:57:16 xjj          1000        0
~~~~~
~
~   SubMenu:
~   1.pay
~   2.inquire the information of your card
~   3.inquire the transactions
~   4.deposit
~   5.transfer out
~   6.repay
~   7.return to the main menu
~
~~~~~
Please enter what you want to do next:
3
operation    amount    time          balance    used_quota  spare_quota
deposit      1000      2019/06/01/14:59:15 1000        0           250

```

选择 1 进行支付，支付 1050 元（此时使用额度）：

```

Please enter what you want to do next:
1
Please enter the amount you will pay:
1050
You have paid it successfully!

```

再次查询：

```

Please enter what you want to do next:
2
id          date          name          balance    used_quota
1621193458  2019/06/01/14:57:16 xjj          0           50

Please enter what you want to do next:
3
operation    amount    time          balance    used_quota  spare_quota
deposit      1000      2019/06/01/14:59:15 1000        0           250
pay          1050      2019/06/01/15:02:46 0            50          200

```

此时已使用额度 50 元，再次充值 1000 元后选择 6 还款：

```

Please enter what you want to do next:
6
You have repayed successfully!

```

```

Please enter what you want to do next:
3
operation    amount    time          balance    used_quota  spare_quota
deposit      1000      2019/06/01/14:59:15 1000        0           250
pay          1050      2019/06/01/15:02:46 0            50          200
deposit      1000      2019/06/01/15:05:18 1000        50          200
repay        -50       2019/06/01/15:05:23 950         0           250

```

选择 7 退回主菜单，选择 3 绑定卡并添加 id18340181 的校园卡并绑定 id 分别为 123、456 的储蓄卡：

```

Please enter what you want to do next:
3
Please enter the id of your card or add your card into this system:
18340181
There is no such id in this system. To add your id in it, please enter your name and your college:
name:
ljj
college:
sdcs
Enter the deposit_card to bind your card(or enter 0 to stop binding):
123
Enter the deposit_card to bind your card(or enter 0 to stop binding):
456
Enter the deposit_card to bind your card(or enter 0 to stop binding):
0

```

进入子菜单:

```

~
~ SubMenu: ~
~ 1.pay ~
~ 2.inquire the information of your card ~
~ 3.inquire the transactions ~
~ 4.deposit ~
~ 5.transfer out ~
~ 6.show binding deposit cards ~
~ 7.repay ~
~ 8.bind deposit_card ~
~ 9.return to the main menu ~
~
~

```

选择 4，对校园卡用刚才已充钱的储蓄卡 1621193458 充值 150 元并查询:

```

Please enter what you want to do next:
4
Transfer in by 1.cash 2. other deposit_card
2
Please enter the id of deposit_card you will transfer from:
1621193458
Please enter the amount you will transfer in:
150
Transfer to 1.my campus_card 2.my deposit_card
1
You have transfered out successfully!
You have deposit successfully!

```

```

Please enter what you want to do next:
2

```

id	date	name	college	balance
18340181	2019/06/01/15:07:11	ljj	sdcs	150

后退回主菜单选择 1 校园卡添加 id 260139，并充值 30 元:

```

3

```


operation	amount	time	balance
transfer_in	30	2019/06/01/15:18:29	30

完全退出该系统后，再次进入对 1621193458 的储蓄卡查询流水账：

```
Please enter what you want to do next:
3
operation      amount      time          balance      used_quota  spare_quota
deposit        1000        2019/06/01/14:59:15 1000         0           250
pay            1050        2019/06/01/15:02:46 0             50          200
deposit        1000        2019/06/01/15:05:18 1000         50          200
repay          -50         2019/06/01/15:05:23 950          0           250
transfer_out   -150        2019/06/01/15:10:02 800          0           250
transfer_out   -30         2019/06/01/15:18:29 770          0           250
```


信息不会丢失，其他功能板块的测试类似。

记录的 txt 文件如下，校园卡信息：

 campus\_card\_inf - 记事本


文件(E)	编辑(E)	格式(O)	查看(V)	帮助(H)
260139	2019/06/01/15:18:20	lhy	sdcs	30

储蓄卡信息：


 deposit\_card\_inf - 记事本

文件(E)	编辑(E)	格式(O)	查看(V)	帮助(H)
1621193458	2019/06/01/14:57:16	xjj	770	0

1621193458 和 260139 的流水账：

 1621193458 - 记事本

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)
deposit	1000	2019/06/01/14:59:15	1000	0 250
pay	1050	2019/06/01/15:02:46	0	50 200
deposit	1000	2019/06/01/15:05:18	1000	50 200
repay	-50	2019/06/01/15:05:23	950	0 250
transfer_out	-150	2019/06/01/15:10:02	800	0 250
transfer_out	-30	2019/06/01/15:18:29	770	0 250

 260139 - 记事本

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)
transfer_in	30	2019/06/01/15:18:29	30	

但是，由于绑定卡的特殊，绑定的储蓄卡数目不定，绑定卡不能实现在退出系统后重新载入本来绑定的储蓄卡，信息会丢失，因此加入另外绑定卡的功能，让用户重新绑定，不过仅限于该类卡。

## 六、设计心得

在这次大作业中，我运用了现阶段所学的类的继承和面向对象的编程思想，我意识在项目开始之前对整个项目的思考是很重要的，最起码要有一个大致的框架，如果边做边想的话，会发现很多和之前写下的矛盾的方面，特别是对于接口而言，会由于不能把握该传递些什么数据和对象而使得整个编程进度减慢。同时，也更加熟悉面向对象的程序设计和对类的继承和虚函数的使用，更加熟悉对链表的应用。还有对文件输入输出的操作学习得到了练习，对于需要存储的数据可以使用文件加链表的方式结合使用，而对于要常写常读且不定的数据，可以采用需时读写的方式，但对于一些不定数目的数据，还是不能很好掌握对其的输入。