

Project 3 实验报告*

17341015 计一陈鸿峥

1 实验目的

开发一个校园电子卡管理系统，可以实现以下三种电子卡的功能

- 校园卡：支付、查询功能，需要注意
 1. 不可透支
 2. 查询内容包括：
 - 支出流水
 - 转入流水
 - 校园卡信息（学号、姓名、学院）
 3. 只支持现金转入，不支持转出和提取
- 储蓄卡：支付、查询、转账功能，需要注意
 1. 支付可以透支一定额度
 2. 查询内容包括：
 - 支出流水
 - 转账流水
 - 储蓄卡信息（卡号、发卡日期、姓名）
 3. 支持转出至三种不同的卡，支持提款，支持现金或其他储蓄卡转入
- 绑定卡：前两者的结合，但以校园卡功能为主，储蓄卡功能为辅

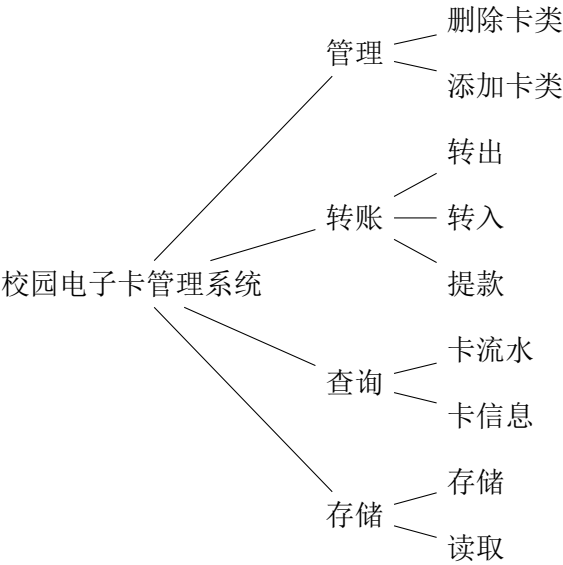
2 实验环境

1. 系统环境：Windows 10
2. 编程环境：Sublime Text 3
3. 编程语言：C++
4. 编译环境：gcc 6.3.0

*本文采用 \LaTeX 进行编写

3 实现思路

校园电子卡管理系统的功能如下：



4 设计细节

4.1 文件关系

本多项式计算器包含六个源程序文件：

1. `main.cpp` 为主文件，包含上述四个模块及具体的交互界面
2. `card.hpp` 包含 `card`基类，实现基础卡的功能
3. `campus_card.hpp` 包含 `campus_card`类，继承 `card`基类，并具有自己独特的功能
4. `deposit_card.hpp` 包含 `deposit_card`类，继承 `card`基类，并具有自己独特的功能
5. `binding_card.hpp` 包含 `binding_card`类，继承 `campus_card`和 `deposit_card`类，并具有自己独特的功能
6. `card_repository.hpp` 包含 `card_repository`类，即存储以上三种卡的仓库

具体依赖关系如下图所示。



Exported from Incocontrol™

4.2 头文件

1. <string>,<vector>为C++标准库的容器，用于存储数据
2. <iostream>,<fstream>用于命令行及文件的输入输出
3. <iomanip>用于控制输出格式
4. <regex>,<iterator>为正则表达式和迭代器，用于字符串的分割

4.3 类结构设计

4.3.1 日期类

由于记录卡流水信息时需要用到日期，故先实现一个日期类，实际上为C的时间库<ctime>的封装
由于本校园卡系统是简单实现，故记录流水时只记录到日，而没有精确到时分秒

```
13 struct Date
14 {
15     int Year;
16     int Month;
17     int Day;
18     void getNowDate()
19     {
20         time_t now = time(0);
21         tm *ltm = localtime(&now);
22         Year = 1900 + ltm->tm_year;
23         Month = 1 + ltm->tm_mon;
24         Day = ltm->tm_mday;
25     }
26     std::string toString() const
27     {
28         std::string resDate;
29         resDate = std::to_string(Year) + "/"
30                 + std::to_string(Month) + "/"
31                 + std::to_string(Day);
32         return resDate;
33     }
34     Date& operator=(const Date& other)
35     {
36         Year = other.Year;
37         Month = other.Month;
38         Day = other.Day;
39         return *this;
40     }
41 };
```

4.3.2 流水类

同样，记录卡的信息时需要记录流水，故直接开一个流水类，实现如下

```

43  struct flow_record
44  {
45      Date date;
46      // 记录流水操作的地点
47      // 如果是转账，则本条会显示对应的卡号；
48      // 否则显示消费地点或取款地点
49      std::string place;
50      // 卡的金额流动
51      // 如果值为正，即为存入或转入操作；
52      // 如果值为负，即为提款或转出操作
53      double moneyIO;
54      // 存储当前余额
55      // 值为正，则卡内还有余额
56      // 值为负，则卡为透支状态
57      double moneyCurr;
58  };

```

4.3.3 卡基类

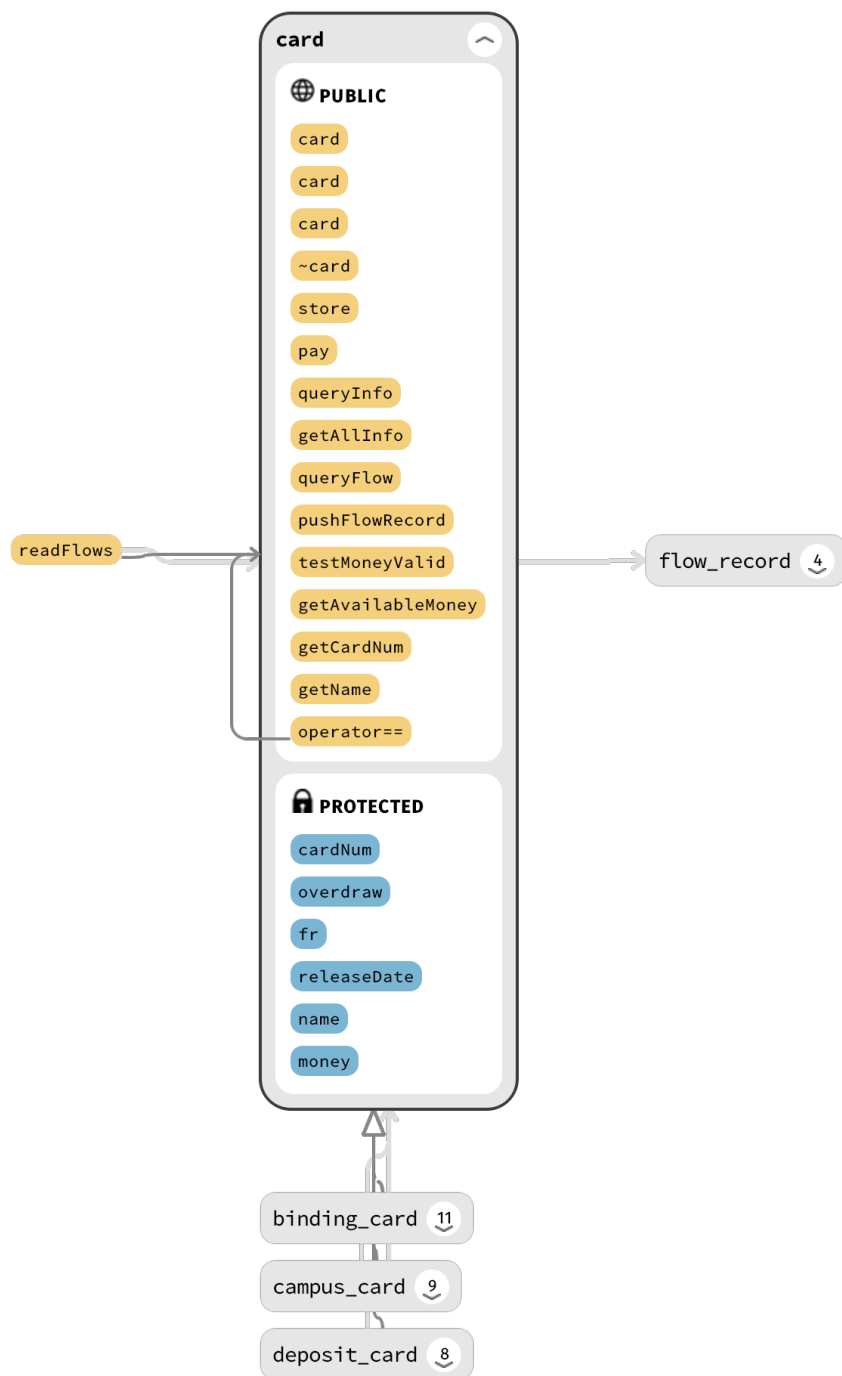
作为其他三类卡的基类，需要将大部分操作实现，具体实施细节如下
 由于基类的成员都需被继承，故设置为 `protected`

```

60 class card{
61 public:
62     // 构造函数
63     card() = default;
64     // 正常的构造函数
65     card(int _cardNum, std::string _name, int _overdraw = 0):
66         cardNum(_cardNum), name(_name), overdraw(_overdraw)
67     {
68         releaseDate.getNowDate();
69     };
70     // 为方便从文件流中读入数据，此处重新实现了一个构造函数，增添了直接读入日期的功能
71     card(int _cardNum, Date _releaseDate, std::string _name, int _overdraw = 0):
72         cardNum(_cardNum), releaseDate(_releaseDate), name(_name), overdraw(_overdraw){};
73
74     //析构函数
75     ~card() = default;
76
77     // 为子类提供存储、支付、查询接口
78     virtual bool store(double _money, const std::string _place) = 0; // pure specifier
79     virtual bool pay(double _money, const std::string _place) = 0;
80     virtual std::vector<std::string> queryInfo() const = 0;
81     // 由于流水信息结构一致，不需每个类单独实现，故本函数没有变为虚函数
82     std::vector<flow_record> queryFlow() const;
83
84     // 方便文件流输出信息
85     virtual std::vector<std::string> getAllInfo() const;
86     // 方便文件流读入历史流水
87     inline void pushFlowRecord(flow_record& _fr);
88
89     // 测试是否能从卡里继续提钱或支付
90     // 主要看当前需要的金额是否大于透支金额与存储金额之和
91     inline bool testMoneyValid(double money) const;
92     // 获得当前可用的金额，包括可透支部分
93     inline double getAvailableMoney() const;
94
95     // 获得卡信息
96     inline int getCardNum() const;
97     inline std::string getName() const;
98
99     // 判断两张卡的持有人是否一致
100     inline bool operator==(const card& other) const;
101
102 protected:
103     // 历史流水
104     std::vector<flow_record> fr;
105     // 基础卡信息
106     int cardNum = 0;
107     Date releaseDate;
108     std::string name;
109     // 当前余额，若为负，即透支
110     double money = 0; // can be negative
111     // 可透支金额，为常数，一旦确定不可修改
112     // 预设可透支金额为0
113     double overdraw = 0; // fixed number
114 };

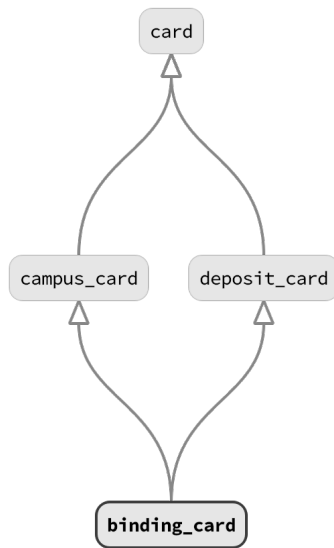
```

卡基类图示如下



Exported from Sourcetrail®

以下三个类与卡基类的关系如图所示



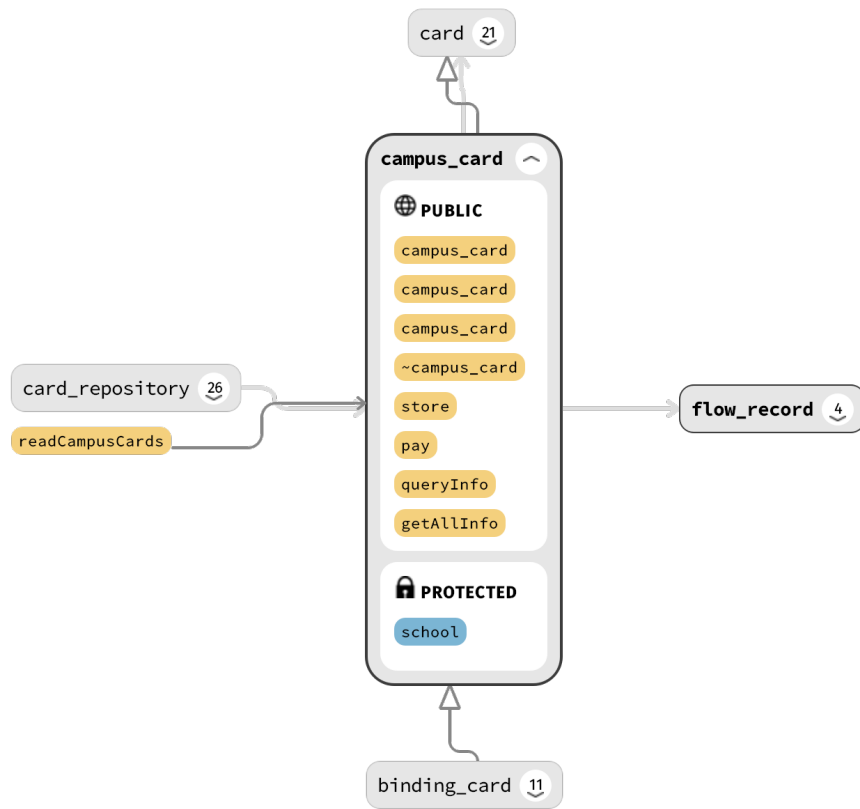
Exported from Sourcetrail™

4.3.4 校园卡类

继承 `card` 基类，注意是虚继承，因之后还会被继承一次，且是菱形继承；若不进行虚继承，会导致二义性错误

```
12 class campus_card : virtual public card{ // 由于要进行菱形继承，故这里采用虚继承
13 public:
14     // 构造函数
15     campus_card() = default;
16     // 标准构造函数
17     // 由于校园卡不可透支，故overflow设为0
18     campus_card(int _cardNum, std::string _name, std::string _school):
19         card(_cardNum, _name, 0), school(_school){};
20     // 方便文件流读入
21     campus_card(int _cardNum, Date _releaseDate, std::string _name, std::string _school):
22         card(_cardNum, _releaseDate, _name, 0), school(_school){};
23
24     // 析构函数
25     ~campus_card() = default;
26
27     // 具体实施存储、支付、查询功能
28     bool store(double _money, const std::string _place);
29     bool pay(double _money, const std::string _place);
30     std::vector<std::string> queryInfo() const;
31     std::vector<std::string> getAllInfo() const;
32
33 protected:
34     // 由于大部分成员已经在基类里实现，故这里只实现校园卡特有的成员，即学院名
35     std::string school;
36 };
```

校园卡类图示如下



Exported from Sourcetrail

4.3.5 储蓄卡类

继承 `card` 基类，注意是虚继承，因之后还会被继承一次，且是菱形继承；若不进行虚继承，会导致二义性错误

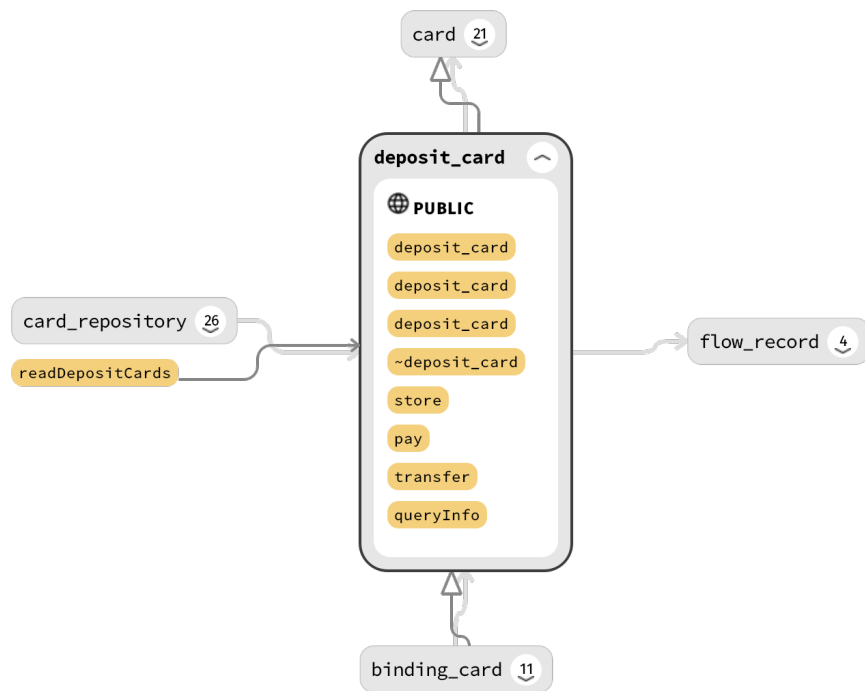
注意转账功能采用了泛型编程的思想


```

12 // 预设储蓄卡透支金额，默认为1000元
13 #define OVERDRAW_LIMIT 1000
14
15 class deposit_card : virtual public card{ // 由于要进行菱形继承，故这里采用虚继承
16 public:
17     // 构造函数
18     deposit_card() = default;
19     // 标准构造函数
20     // 由于校园卡不可透支，故overflow设为0
21     deposit_card(int _cardNum, std::string _name, double _overdraw = OVERDRAW_LIMIT):
22     {
23         card(_cardNum, _name, _overdraw){};
24     }
25     // 方便文件流读入
26     deposit_card(int _cardNum, Date _releaseDate, std::string _name):
27     {
28         card(_cardNum, _releaseDate, _name, 0){};
29     }
30
31     // 析构函数
32     ~deposit_card() = default;
33
34     // 具体实施存储、支付、查询功能
35     bool store(double _money, const std::string _place);
36     bool pay(double _money, const std::string _place);
37     std::vector<std::string> queryInfo() const;
38
39     // 转账功能
40     // 由于转账可以转给三种不同的卡，故这里直接输入一个card类（泛型编程的思想）
41     bool transfer(card& camp, double _money); // genetic
42
43 protected:
44     // 由于基本信息已在基类中实施，故本类没有自己的成员
45 };

```

储蓄卡类图示如下



Exported from Sourcetrail™

4.3.6 绑定卡类

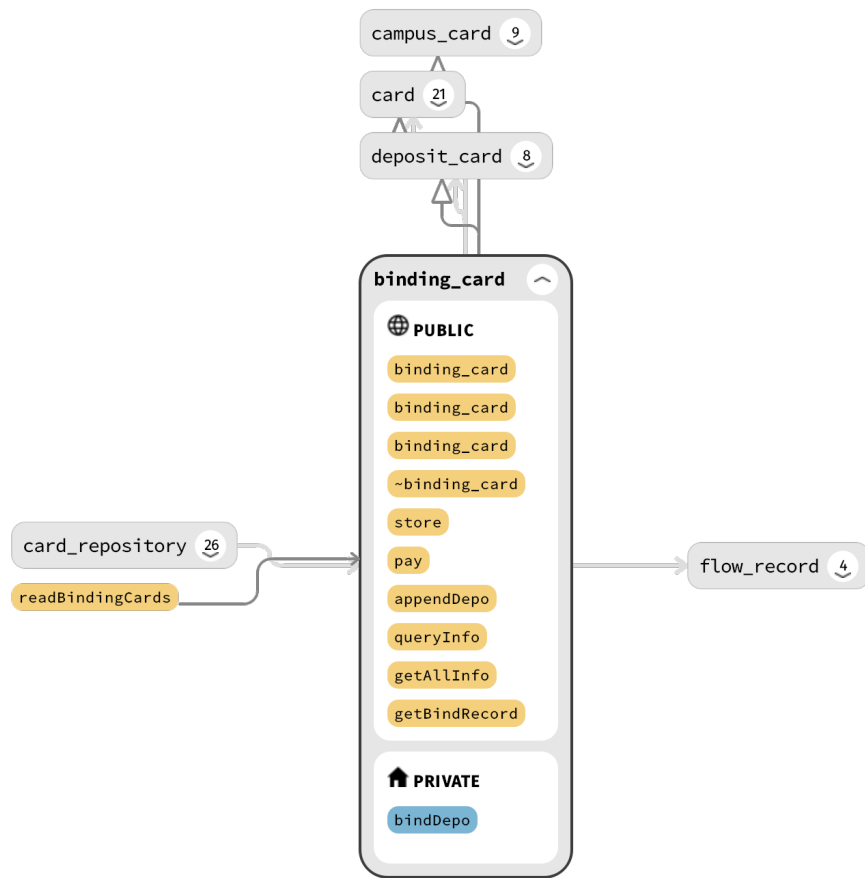
继承 `campus_card`和 `deposit_card`类，同时将绑定的储蓄卡信息以指针的形式存入类成员中

```

13  class binding_card : public campus_card, public deposit_card{ // 菱形继承
14  public:
15      // 构造函数
16      binding_card() = default;
17      // 标准构造函数
18      // 由于主要功能为校园卡，故与校园卡的构造相同
19      binding_card(int _cardNum, std::string _name, std::string _school):
20      |   campus_card(_cardNum, _name, _school){};
21      // 方便文件流读入
22      binding_card(int _cardNum, Date _releaseDate, std::string _name, std::string _school):
23      |   campus_card(_cardNum, _releaseDate, _name, _school){};
24
25      // 析构函数
26      ~binding_card() = default;
27
28      // 具体实施存储、支付、查询功能
29      bool store(double _money, const std::string _place);
30      bool pay(double _money, const std::string _place);
31      std::vector<std::string> queryInfo() const;
32      std::vector<std::string> getAllInfo() const;
33
34      // 添加绑定的储蓄卡
35      bool appendDepo(deposit_card* depo);
36      // 获得绑定的储蓄卡号
37      std::vector<int> getBindRecord() const;
38
39  private:
40      // 由于本类在卡仓库类之前实施，故只能存储绑定储蓄卡的指针
41      // 而不能储存其在仓库内的位置，否则会造成交叉引用错误
42      std::vector<deposit_card*> bindDepo;
43  };

```

绑定卡类图示如下



Exported from Sourcetrail™

4.3.7 卡仓库类

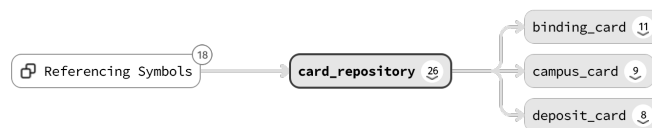
统一存储三种卡，为主函数提供接口

```

12 class card_repository{
13 public:
14     // 构造函数与析构函数
15     card_repository() = default;
16     ~card_repository() = default;
17
18     // 添加卡
19     void appendDepositCard(const std::string name);
20     void appendCampusCard(const std::string name, const std::string school);
21     void appendBindingCard(const std::string name, const std::string school, std::vector<int> cardli);
22     // 方便文件流读入, 先主函数中构建好卡信息后再读入仓库
23     void appendDepositCard(deposit_card& dc);
24     void appendCampusCard(campus_card& cc);
25     void appendBindingCard(binding_card& bc, std::vector<int> cardli);
26
27     // 转账、提取、存储
28     bool transferMoney(int numFrom, int numTo, double money);
29     bool withdrawMoney(int numCard, double money);
30     bool depositMoney(int numCard, double money);
31
32     // 删除卡
33     bool deleteCard(int num);
34
35     // 在仓库中查询卡
36     bool findName(const std::string name);
37     int QType(int num);
38     campus_card* findCamp(int num);
39     deposit_card* findDepo(int num);
40     binding_card* findBind(int num);
41     inline int getTotal() const { return totalNum; };
42     inline std::vector<deposit_card> getDepoCard() const { return depoCard; };
43     inline std::vector<campus_card> getCampCard() const { return campCard; };
44     inline std::vector<binding_card> getBindCard() const { return bindCard; };
45
46 private:
47     // 存储三类卡的全部信息
48     std::vector<deposit_card> depoCard;
49     std::vector<campus_card> campCard;
50     std::vector<binding_card> bindCard;
51     // 存储卡的类型
52     // 由卡号映射到{1,2,3}
53     // 其中, 1为校园卡, 2为储蓄卡, 3为绑定卡
54     std::map<int,int> cardType;
55     // 开卡时的卡号, 每次开一张卡, 总数目就加1
56     // 但注意删除卡不会减1, 以防冲突
57     int totalNum = 0;
58 };

```

卡仓库类图示如下



Reported from Sourcecraft!

4.4 主函数设计

主函数中实现的函数如下, 大多为交互界面的输入输出以及调用卡仓库类的接口

```

21 // Module
22 void manageCards(card_repository& repo);
23 void tranCards(card_repository& repo);
24 void queryCards(card_repository& repo);
25 void readAllCards(card_repository& repo);
26 void saveAllCards(card_repository& repo);
27 // Management
28 void appendCampusCard(card_repository& repo);
29 void appendDepositCard(card_repository& repo);
30 void appendBindingCard(card_repository& repo);
31 void deleteCard(card_repository& repo);
32 // Transaction
33 void withdrawMoney(card_repository& repo);
34 void depositMoney(card_repository& repo);
35 void transferMoney(card_repository& repo);
36 // Query
37 void queryCardInfo(card_repository& repo);
38 void queryFlowRecord(card_repository& repo);
39 // Others
40 void show_main_manual();
41 void interface();
42 vector<string> split(const string& input, const string& regex);
43 bool validNameTest(const string& name);
44 bool vecLenTest(vector<string> vec,int len);

```

4.5 交互界面

由于交互界面实现代码过长，故不在此贴图，请到 main.cpp 中查看

```

=====
- Reddie's Campus Card Management System! -
=====
Command List:
1. Card management
2. Transaction
3. Query
4. Exit
Note:1. To append, delete a card, please enter "1".
     2. To withdraw, transfer, deposit money, please enter "2".
     3. To query card information and flow record, please enter "3".
     4. To save records, you SHOULD exit by entering "4".
Please enter the number of operation. Enter "4" to quit.
>>>

```

- 用<iomanip>控制输出格式确保美观

- 用`cin.clear()`和`cin.sync()`清空缓存区，确保每次输入被程序读入后不会有残留，避免干扰下一次操作
- 当字符串长度为0或大于1时循环读入字符串，避免读入空行或者错误输入
- 多重输入判定，对输入名字的合法性、名字是否与已有的卡冲突等进行判定，确保输入准确合法

4.6 输入输出及存储

1. 输出可在命令行中手动输入，或者从文件流中直接读入
2. 输出同时有命令行的输出和文件流的输出（但要选择保存），输出格式同输入
3. 具体操作在交互界面均会提示，按照提示要求进行即可

由于存储的内容相似，故这里采用了泛型编程的思想，构造了一个模板，简化代码

```

339 template <class Data>
340 void saveCards(Data& data, ofstream& outfile)
341 {
342     outfile << data.size() << endl;
343     for(auto pdata = data.begin(); pdata != data.end(); ++pdata)
344     {
345         vector<string> temp = pdata->getAllInfo();
346         for (auto ptemp = temp.begin(); ptemp != temp.end(); ++ptemp)
347             { outfile << *ptemp << " ";}
348         outfile << endl;
349         vector<flow_record> fr = pdata->queryFlow();
350         outfile << fr.size() << endl;
351         for (auto ptemp = fr.begin(); ptemp != fr.end(); ++ptemp)
352             outfile << (ptemp->date).Year << " " << (ptemp->date).Month << " " << (ptemp->date).Day << " "
353                 << ptemp->place << " " << ptemp->moneyIO << " " << ptemp->moneyCurr << endl;
354     }
355 }

```

存储会分别存在三个文档，`Campus_card.dat`、`Deposit_card.dat`，`Binding_card.dat`中，存储结构类似下图

```

1  2
2  2 2018 5 19 HYJ
3  2
4  2018 5 19 Withdraw -66 -66
5  2018 5 19 Store 30 -36
6  3 2018 5 19 CHZ
7  1
8  2018 5 19 Store 50 50

```

第一行为一个数字 n ，代表该种卡有 n 张

接下来一行会是第一张卡的信息，按顺序依次是卡号、开卡日期、姓名（学院）

第三行是一个数字 m ，代表有 m 个流水数据

流水数据依次按照日期、地点、变化金额、余额来记录

如此往复，直到所有 n 条数据全部存储完毕

对于绑定卡，则还会添加一行卡号，对应绑定的储蓄卡号码

4.7 值得一提的细节

1. 代码符合C++11标准，尽可能在易读代码的同时简化代码
2. 对于类的实施，一定会在必要的地方加上 `const`和 `inline`等关键词修饰
3. 对于指针、引用的使用及其小心，尽可能避免多余、意外操作

5 程序测试及实验结果

添加卡类效果如下

```
Command List:
1. Create a new campus card
2. Create a new deposit card
3. Create a new binding card
4. Cancel a card
5. Back

Please enter the number of operation. Enter "5" to return.
>>> 1

Please enter your name and school, sperating them by space.
Note: Your name should not contain space, which means you need to use caption to distinguish your first name and family name.
So is school name. You can only contain letters, hyphen or underline.
>>> HJ SDCS
Append successfully. Your card number is #1.
Please enter the number of operation. Enter "5" to return.
>>> 2

Please enter your name, which should not contain space, meaning you need to use caption to distinguish your first name and family name.
Note: Default overdraw limit is 1000 yuan. You can change it by calling the administrator.
>>> CHZ
Append successfully. Your card number is #2.
Please enter the number of operation. Enter "5" to return.
>>>
```

转账功能如下

```
Command List:
1. Transfer
2. Withdraw
3. Deposit
4. Back

Please enter the number of operation. Enter "4" to return.
>>> 1

Please enter your card number, the number of the card you want to transfer money to, and amount of money, sperating them by space.
>>> 1 2 30
Transfer successfully!
Please enter the number of operation. Enter "4" to return.
>>> 3

Please enter your card number and amount of money you want to withdraw, sperating them by space.
>>> 1
Error: You should enter two numbers.
>>> 1 60
Deposit successfully!
Please enter the number of operation. Enter "4" to return.
>>>
```

查询功能如下，可以见到若没有数据，也可以正常输出

```
Command List:
1. Card info
2. Flow record
3. Back

Please enter the number of operation. Enter "3" to return.
>>> 1

Please enter your card number.
>>> 1
1 SDCS HJ
Please enter the number of operation. Enter "3" to return.
>>> 2

Please enter your card number.
>>> 2
Ops...There's no flow record.
```

实现了项目要求的所有功能，写了1000+行代码考虑了各种极端情况，确保程序足够鲁棒。以下是用到的一些C++11特性：

1. 利用泛型编程的思想，写模板函数，减少代码的重复
2. `split`函数在进行正则匹配时常常会导致程序崩溃：
加上`try, catch`特性捕获并避免发生异常
3. 必要的地方都加上`const`限定，防止误操作修改类的私有数据
4. 使用`auto`自动推断类型
5. 使用`for`的范围语句简化代码
6. 使用智能指针`nullptr`