课 程 设 计 报 告

**课程名称 计算机程序设计基础（2）**

**班 级 无36**

**学 号 2013011189**

**姓 名 郭一隆**

**指导助教**

**2014年7月21日**

1. 系统需求分析

图书管理系统作为管理系统，应具有用户及登录功能。用户种类包括管理员和读者。不同用户具有的权限不同，登入学生成绩管理系统后进行不同操作。详细功能如下表1.1：

|  |  |  |
| --- | --- | --- |
| **用户种类** | **一级功能** | **具体功能** |
| 共有 | 个人中心 | 修改密码 |
| 退出登录 |
| 退出系统 | 退出系统 |
| 管理员 | 用户管理 | 查看所有用户 |
| 新用户注册 |
| 删除用户 |
| 书籍管理 | 查看所有书籍 |
| 新书上架 |
| 旧书下架 |
| 读者 | 借阅书籍 | 检索书籍 |
| 查看馆长推荐 |
| 电子资源 |
| 归还书籍 | 还书 |
| 借阅历史 | 查看借阅历史 |

表1.1

功能具体需求：

* 1. 登录界面：用户从键盘输入用户名和密码，检验是否正确，返回相应信息。登录成功则进入用户菜单界面。
  2. 修改密码：输入原密码，新密码重复输入两次，检验原密码是否符合，新密码是否符合一定规则（如6~15位）。
  3. 退出登录：退回至用户登录界面。
  4. 退出系统：关闭程序。
  5. 新用户注册：管理员选定新用户的身份，新ID号由系统计算得出（大于当前所有ID号），新密码随机生成。
  6. 删除用户：输入ID，检验ID是否存在，存在则删除该用户数据（管理员不得删除管理员）。
  7. 查看所有用户：显示所有用户的基本信息。
  8. 查看所有书籍：显示所有书籍的信息，包括基本信息（ISBN号，同书索引号，书名，作者，出版社）、是否在架上、借阅历史等。
  9. 新书上架：输入书籍的基本信息，添加新书至馆藏。
  10. 旧书下架：撤下旧书，若被正被借出则无法下架。
  11. 检索书籍：按书名或ISBN检索书籍。
  12. 馆长推荐：为读者推荐书籍。考虑在登录首页滚动播出。
  13. 电子资源：跳转至网页。
  14. 还书：归还书籍，若逾期，则扣除一定信用积分；按时还书奖励一定信用积分。所借书籍的借期长短与信用积分有关。
  15. 借阅历史：查看借过的书籍。

1. 总体设计

图书管理系统主要模块：登录模块、菜单模块、用户管理模块、书籍管理模块、读者预约借还模块（附信用系统）。

登录模块：加载登录界面前将所有用户数据从文件读入，以全局数组的形式保存，获取用户输入的信息，与全局数组比对验证。

菜单模块：通过调用<windows.h>中的函数改变控制台字体颜色并实现方向键控制菜单选项。

用户管理模块：查看用户信息，增减用户，实时更新文件。

书籍管理模块：查看书籍信息，增减书籍，实时更新文件。

读者预约借还模块：读者拥有一定初始信用积分，信用积分影响借期长短。读者试图借阅某书时，若在架上则借阅成功；若被借出则预约成功；若被预约则预约失败。另外需考虑预约权也具有有效期，超时作废。按时还书奖励信用积分，超时还书扣除信用积分。

预约借还

用户管理

书籍管理

登录

菜单

学生成绩管理系统

图2.1 系统功能模块图

1. 详细设计

图书管理系统中7个类/结构体的层次图如下图：

Info

Book

Administrator

派 生

管理菜单链表

MenuNode

登录验证

Token

Reader

User

图3.1 类/结构体的层次图

* 1. 登录：从键盘输入获取登录令牌（Token）并通过Token::auth()方法验证，返回User\*基类指针。输入密码时键入的字符以‘\*’回显，支持退格键及左右方向键，遇到不合法的输入则输出‘\a’发出alert。
  2. 菜单界面：本系统亮点之一，显示菜单函数MenuNode::show(int k)每行打印一条菜单选项，其中第k条高亮打印。另一方面，利用getch()非缓冲地读入用户的方向键指令，通过调用自定义的清屏函数（系统清屏+打印系统标题）ClearScreen()，并更改高亮显示的选项，以达到仿图形化菜单的视觉效果及用户体验。
  3. 用户管理：键盘读入新用户相应信息，并以std::ios::app方式存至文件，再根据文件更新全局数组。删除则从全局数组中删除相应用户，再根据全局数组更新文件。
  4. 修改密码：在原密码验证通过的前提下，将新密码存入全局数组，紧接着更新文件。密码保存读取有相应的加密解密函数（自定义的加密算法，安全性很弱），非明文保存密码。
  5. 查看所有书籍：打印所有书籍的基本信息和借阅关系。
  6. 增减书籍：对全局数组进行操作，并更新文件。
  7. 检索书籍：键盘输入关键词或ISBN，至于是ISBN还是书名检索，交给系统自动判断。显示匹配的书籍基本信息，以菜单形式检阅检索结果，ENTER或右方向可进入借阅该书的界面。
  8. 馆长推荐：显示馆长推荐书籍（热门书籍）的基本信息，以菜单形式检阅列表，ENTER或右方向可进入借阅该书的界面。
  9. 电子资源：打印 <http://lib.tsinghua.edu.cn> 请读者自行寻找电子资源。
  10. 归还书籍：显示当前持有的书籍，以菜单形式检阅，ENTER或右方向可进入归还该书的界面。按时还书奖励信用积分，逾期还书扣除信用积分。
  11. 借阅历史：显示所有曾借过的书籍，包括起止时间。
  12. 预约：某些借阅不成功的情况自动转为预约，预约期为3天（上一位读者还书日起）。
  13. 信用积分：信用积分 < 0无法借书，借期长短与信用积分正相关。

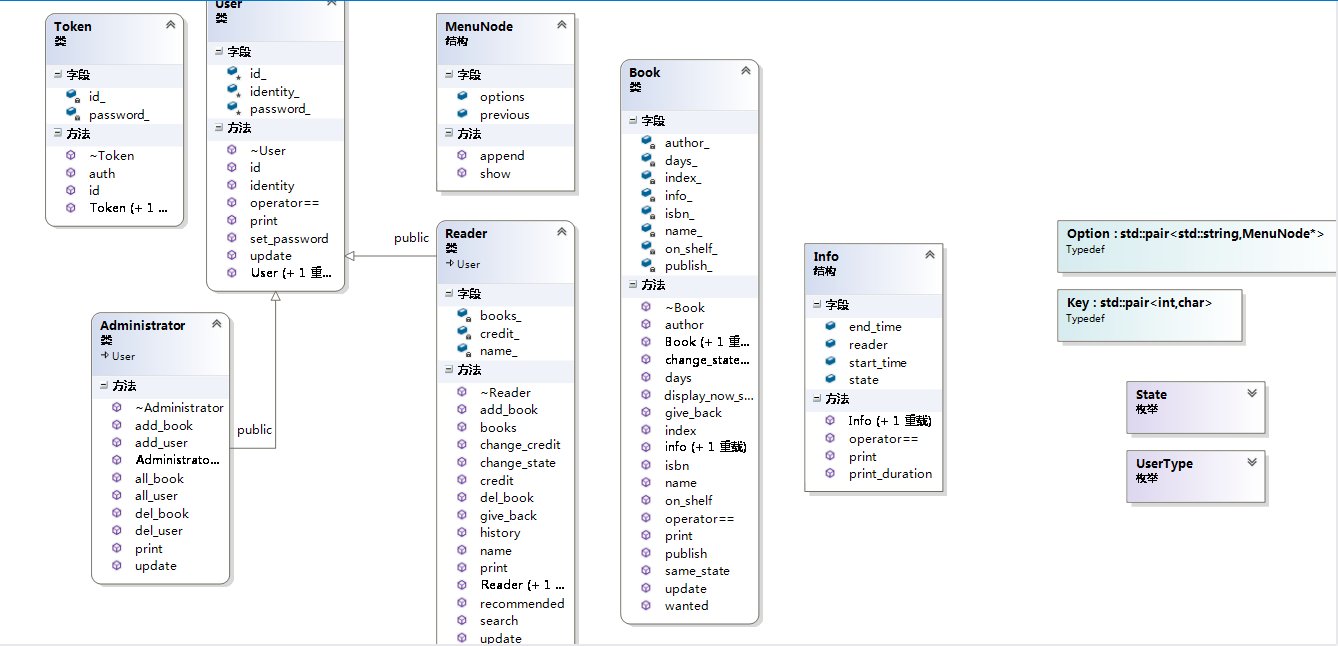


图3.1 UML缩略图

1. 系统调试

图书管理系统的实现与学生成绩管理系统的实现十分相似。学生成绩管理系统曾出现的错误此次已吸取教训，减少了很多。

将声明部分放入\*.h，函数定义实现放入\*.cpp。编译链接中出现的错误包括粗心导致的变量拼写错误；头文件包含顺序混乱； STL容器中迭代器iterator和const\_iterator的用法错误；汉字编码问题；全局变量的声明（extern）和定义；缺少相应类型的重载函数；缺少forward declaration等。

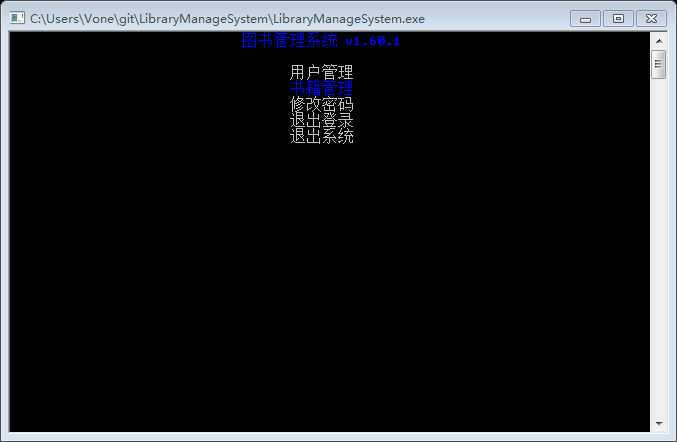
程序编译通过后继续调试，出现的问题包括：许多界面显示结果后直接跳转，导致用户无法看清信息，适时地加入getch()非缓冲不回显的读字符函数用以停留界面；std::cin读取失败后未调用std::cin.sync()清空缓存；密码退格依然以‘\*’回显，加入判断getch()是否为‘\b’并打印相应数量的‘\b’和‘ ’得到解决，之后又加入对左右方向键的相应，密码输入至一半可以利用方向键调整光标位置并保证读入的密码与用户输入意图相同；读取文件到达文件尾导致的错误；用户的不合法输入导致程序进入死循环，原因：实现菜单选项需要获取用户的方向键输入，而一个方向键需要getch()两次，第一次getch()值为224（windows平台），第二次getch()为72,80,75,77（上下左右分别对应），而普通字符只需getch()一次，再getch()第二次则会等待用户输入下一字符，为解决此问题，typedef std::pair<int, char> Key作为新的用户按键类型，自定义MyGetCh()函数利用<windows.h>中的kbhit()函数自动判断应该getch()的次数times，并获取到最后一次getch()的char型值，返回std::make\_pair(times, char)，用MyGetCh()替换原程序中的getch()，成功地解决了用户不合法输入程序不能正常运行的问题，极大地增强了程序的鲁棒性。这些问题与学生成绩管理系统的问题相同，一并解决。

1. 测试结果与分析

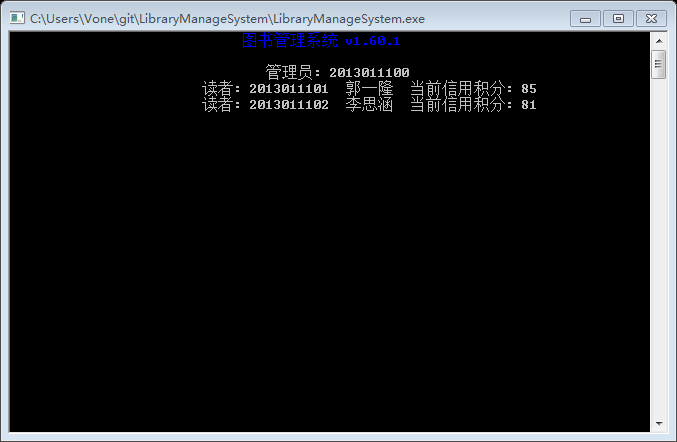
程序运行截图：



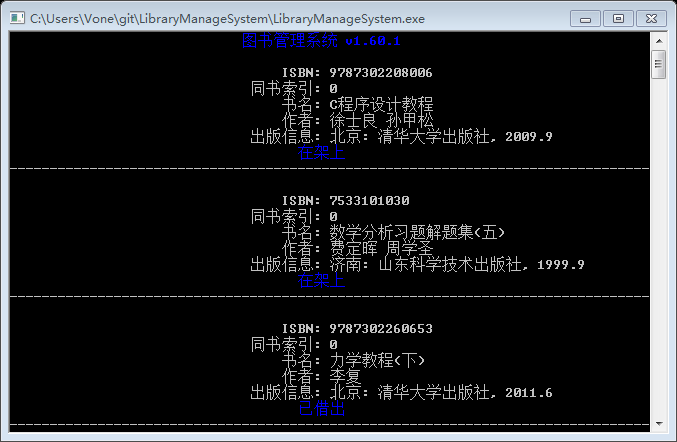
登录界面



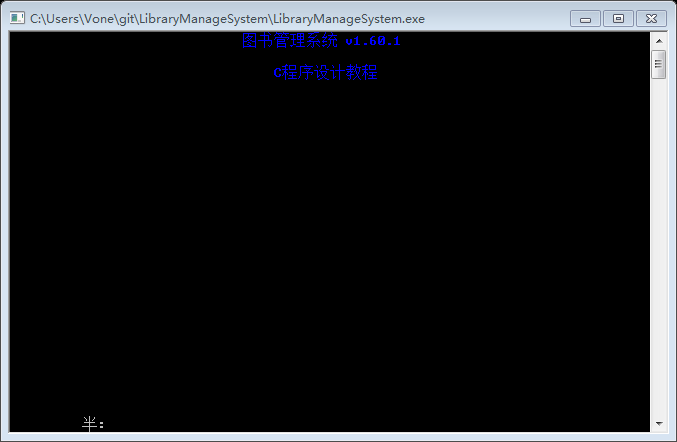
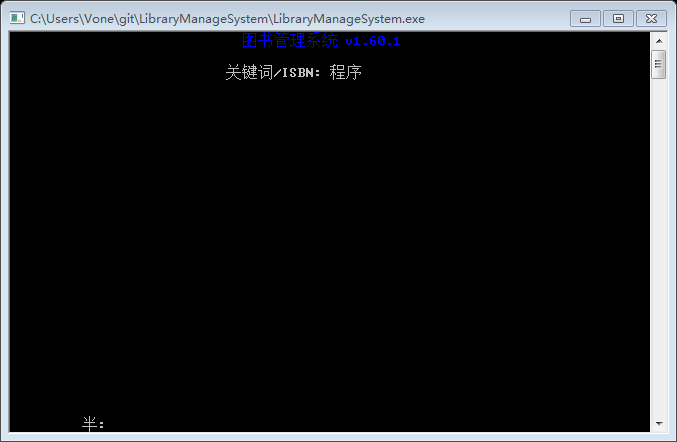
菜单界面



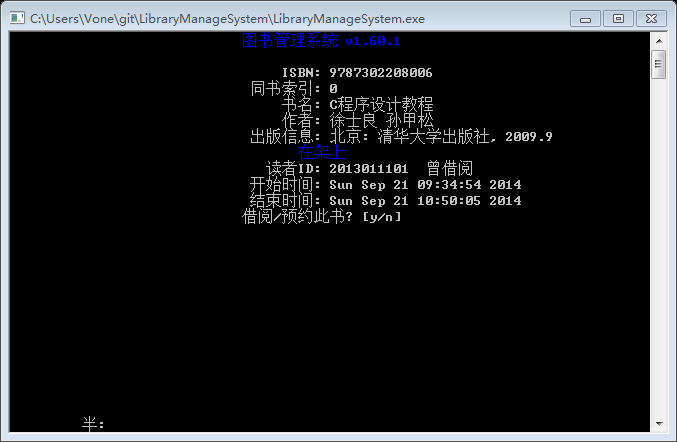
所有用户



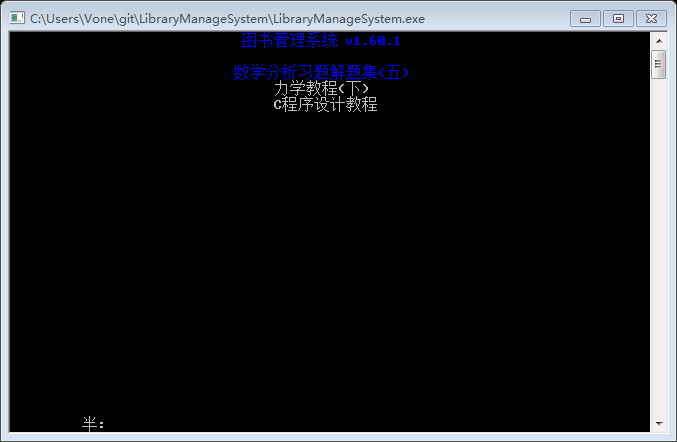
所有书籍



检索书籍及检索结果



借阅/预约书籍



馆长推荐



自助还书



新用户

程序特色：

* 1. 密码‘\*’回显，支持退格和方向键，识别F1~F12和全角字符并自动过滤。
  2. 菜单选项高亮显示，接受方向键、回车键和ESC键，其他无效输入自动忽略，调用清屏函数刷新菜单实现舒适的视觉效果。
  3. 显示书籍列表结果直接作为菜单，可进行选择，并借阅/预约/归还。
  4. 密码非明文保存。新用户生成随机密码。
  5. 程序界面居中显示。尤其是菜单居中显示，较美观。

程序不足：

a) 除了登录界面，后续的操作界面未显示当前操作的用户信息。

b) 居中有时无法完全对齐。造成奇怪的显示状态。不够还凑合，大体是居中的。

c) 密码加密算法安全性弱。

d) 没有相应机制防止读者长期不还书。

e) 电子资源没有实现。

总结

第二个课设作业与学生成绩管理系统很像，程序框架都可以拿过来直接用，函数也是之前实现过的，学生成绩管理系统给图书管理系统作了良好的铺垫。不过bug是不可能没有的，调试过程中，还是有缺分号少括号这种让人哭笑不得的低级错误，也有“常量中有换行符”这种让人不知所云逼格极高的编码问题，不过可喜的是，学生成绩管理系统为我积累下充足的debug经验，问题迎刃而解。第二个程序也完成的更加成熟，更有成就感。

1. 附录：源程序清单

// 先贴上主函数，之后头文件源文件按文件名字母顺序依次贴上

// 附几组用户名和密码

// 管理员 2013011100:123456

// 读者 2013011101:123456

// 读者 2013011102:lisihan

// main.cpp

#include "global.h"

#include "token.h"

#include "menu.h"

#include "file.h"

#include <conio.h>

#include <map>

int main(int argc, char const \*argv[])

{

login: // 登录标签

ClearScreen();

ReadAll();

Token tmp;

User\* user;

while (true)

{

tmp = Login();

if (tmp.auth() != -1)

{

user = users[Find(users, tmp.id())];

MediatePrint("登录成功!\n");

MediatePrint("欢迎 \n");

user->print();

MyGetCh();

break;

}

else

{

ClearScreen();

MediatePrint("学号(工号)或密码错误! 重新输入? \n");

MediatePrint("[回车继续, q退出系统] ");

Key tmp = MyGetCh();

if (tmp == std::make\_pair(1, 'q') || tmp == std::make\_pair(1, 'Q'))

{

HighlightPrint("\n再见!\n");

return 0;

}

ClearScreen();

}

}

ClearScreen();

switch (user->identity())

{

// 管理员菜单

case ADMINISTRATOR:

{

Administrator admin = admins[Find(admins, tmp.id())];

MenuNode top\_menu, user\_menu, book\_menu;

top\_menu.previous = user\_menu.previous = book\_menu.previous = &top\_menu;

top\_menu.append(Option("用户管理", &user\_menu));

top\_menu.append(Option("书籍管理", &book\_menu));

top\_menu.append(Option("修改密码", NULL));

top\_menu.append(Option("退出登录", NULL));

top\_menu.append(Option("退出系统", NULL));

user\_menu.append(Option("所有用户", NULL));

user\_menu.append(Option("注册新用户", NULL));

user\_menu.append(Option("删除用户", NULL));

book\_menu.append(Option("所有书籍", NULL));

book\_menu.append(Option("新书上架", NULL));

book\_menu.append(Option("旧书下架", NULL));

MenuNode now = top\_menu;

int k = 0;

while (true)

{

now.show(k);

Key ch = MyGetCh();

if (ch == UP)

{

if (--k < 0)

k = now.options.size() - 1;

continue;

}

if (ch == DOWN)

{

if (++k > now.options.size() - 1)

k = 0;

continue;

}

if (ch == LEFT || ch == ESCAPE)

{

now = \*now.previous;

k = 0;

continue;

}

if (ch == RIGHT || ch == ENTER)

{

if (now.options[k].second == NULL)

{

std::map<std::string, int> mp;

mp["修改密码"] = 0;

mp["退出登录"] = 1;

mp["退出系统"] = 2;

mp["所有用户"] = 3;

mp["注册新用户"] = 4;

mp["删除用户"] = 5;

mp["所有书籍"] = 6;

mp["新书上架"] = 7;

mp["旧书下架"] = 8;

switch (mp[now.options[k].first])

{

case 0: admin.set\_password(); break;

case 1: goto login;

case 2: Exit(); break;

case 3: admin.all\_user(); break;

case 4: admin.add\_user(); break;

case 5: admin.del\_user(); break;

case 6: admin.all\_book(); break;

case 7: admin.add\_book(); break;

case 8: admin.del\_book(); break;

}

k = 0;

continue;

}

else

{

now = \*now.options[k].second;

k = 0;

continue;

}

}

}

}

// 读者菜单

case READER:

{

Reader reader = readers[Find(readers, tmp.id())];

MenuNode top\_menu, checkout\_menu;

top\_menu.previous = checkout\_menu.previous = &top\_menu;

top\_menu.append(Option("借阅书籍", &checkout\_menu));

top\_menu.append(Option("自助还书", NULL));

top\_menu.append(Option("借阅历史", NULL));

top\_menu.append(Option("修改密码", NULL));

top\_menu.append(Option("退出登录", NULL));

top\_menu.append(Option("退出系统", NULL));

checkout\_menu.append(Option("查询书籍", NULL));

checkout\_menu.append(Option("馆长推荐", NULL));

checkout\_menu.append(Option("电子资源", NULL));

MenuNode now = top\_menu;

int k = 0;

while (true)

{

now.show(k);

Key ch = MyGetCh();

if (ch == UP)

{

if (--k < 0)

k = now.options.size() - 1;

continue;

}

if (ch == DOWN)

{

if (++k > now.options.size() - 1)

k = 0;

continue;

}

if (ch == LEFT || ch == ESCAPE)

{

now = \*now.previous;

k = 0;

continue;

}

if (ch == RIGHT || ch == ENTER)

{

if (now.options[k].second == NULL)

{

std::map<std::string, int> mp;

mp["自助还书"] = 0;

mp["借阅历史"] = 1;

mp["修改密码"] = 2;

mp["退出登录"] = 3;

mp["退出系统"] = 4;

mp["查询书籍"] = 5;

mp["馆长推荐"] = 6;

mp["电子资源"] = 7;

switch (mp[now.options[k].first])

{

case 0: reader.give\_back(); break;

case 1: reader.history(); break;

case 2: reader.set\_password(); break;

case 3: goto login;

case 4: Exit(); break;

case 5: reader.search(); break;

case 6: reader.recommended(); break;

case 7: EBook(); break;

}

k = 0;

continue;

}

else

{

now = \*now.options[k].second;

k = 0;

continue;

}

}

}

}

}

return 0;

}

// book.cpp

#include "book.h"

#include "global.h"

#include "file.h"

#include <conio.h>

std::ifstream &operator >>(std::ifstream &in, Info &info)

{

time\_t now = time(NULL);

int state;

in >> info.reader;

in >> state;

in >> info.start\_time;

in >> info.end\_time;

if (now > info.end\_time)

{

switch (state)

{

case HOLDING: info.state = OVERDUE; break;

case REQUEST: break; // Book::wanted()处理

case WANTED: info.state = OVERDUE; break;

case OVERDUE: info.state = OVERDUE; break;

case RETURNED: info.state = RETURNED; break;

}

}

else

info.state = state;

return in;

}

std::ofstream &operator <<(std::ofstream &out, const Info &info)

{

out << info.reader << '\n';

out << info.state << '\n';

out << info.start\_time << '\n';

out << info.end\_time << '\n';

return out;

}

void Info::print()

{

std::cout << std::setw(WIDTH / 2) << "读者ID: " << reader;

switch (state)

{

case HOLDING:

{

std::cout << " 借阅中\n";

break;

}

case REQUEST:

{

std::cout << " 已预约\n";

break;

}

case WANTED:

{

std::cout << " 借阅中, 被预约, 借期缩短\n";

break;

}

case OVERDUE:

{

std::cout << " 逾期未换, 受到惩罚\n";

break;

}

case RETURNED:

{

std::cout << " 曾借阅\n";

break;

}

default: break;

}

print\_duration();

return;

}

void Info::print\_duration()

{

// struct tm \* time\_start, \* time\_end;

// time\_start = localtime(&start\_time);

// time\_end = localtime(&end\_time);

std::cout << std::setw(WIDTH / 2) << "开始时间: " << ctime(&start\_time);

std::cout << std::setw(WIDTH / 2) << "结束时间: " << ctime(&end\_time);

return;

}

Book::Book(std::string isbn, std::string name, std::string author,

std::string publish, int days):

isbn\_(isbn), name\_(name), author\_(author),

publish\_(publish), index\_(0), on\_shelf\_(true), days\_(days)

{

for (std::vector<Book>::iterator it = all\_books.begin(); it != all\_books.end(); ++it)

{

if (it->isbn() == isbn)

++index\_;

}

}

std::ifstream &operator >>(std::ifstream &in, Book &book)

{

in >> book.isbn\_;

in.get();

getline(in, book.name\_);

getline(in, book.author\_);

getline(in, book.publish\_);

in >> book.index\_;

in >> book.on\_shelf\_;

in >> book.days\_;

char start\_flag;

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

Info tmp;

in >> tmp;

book.info\_.push\_back(tmp);

}

}

}

else

in.seekg(-1, std::ios::cur);

return in;

}

std::ofstream &operator <<(std::ofstream &out, const Book &book)

{

out << book.isbn\_ << '\n';

out << book.name\_ << '\n';

out << book.author\_ << '\n';

out << book.publish\_ << '\n';

out << book.index\_ << '\n';

out << book.on\_shelf\_ << '\n';

out << book.days\_ << '\n';

out << "\*\n";

for (std::vector<Info>::const\_iterator it = book.info\_.begin(); it != book.info\_.end(); ++it)

out << \*it;

out << "#\n\n";

return out;

}

Info &Book::info(int reader\_id)

{

for (std::vector<Info>::iterator it = info\_.begin(); it != info\_.end(); ++it)

{

if (it->reader == reader\_id)

return \*it;

}

}

std::vector<Info> Book::same\_state(int state)

{

std::vector<Info> rst;

for (std::vector<Info>::iterator it = info\_.begin(); it != info\_.end(); ++it)

{

if (it->state == state)

rst.push\_back(\*it);

}

return rst;

}

void Book::change\_state(int reader\_id, int state)

{

info(reader\_id).state = state;

update();

}

void Book::display\_now\_state()

{

for (std::vector<Info>::iterator it = info\_.begin(); it != info\_.end(); ++it)

it->print();

return;

}

void Book::give\_back(int reader\_id)

{

time\_t now = time(NULL);

if (info(reader\_id).state == OVERDUE)

{

Reader tmp = readers[Find(readers, reader\_id)];

int diff = - int(pow(2, (now - info(reader\_id).end\_time) / SECONDS\_A\_DAY));

tmp.change\_credit(diff);

tmp.change\_state(\*this, RETURNED);

info(tmp.id()).end\_time = now;

tmp.update();

HighlightPrint("归还成功, 已超过期限, 信用降低!\n");

}

else

{

Reader tmp = readers[Find(readers, reader\_id)];

tmp.change\_credit(1);

tmp.change\_state(\*this, RETURNED);

info(tmp.id()).end\_time = now;

tmp.update();

HighlightPrint("按时还书成功, 信用提高!\n");

}

on\_shelf\_ = true;

change\_state(reader\_id, RETURNED);

return;

}

void Book::print()

{

std::cout << std::setw(WIDTH / 2) << "ISBN: " << isbn\_ << std::endl;

std::cout << std::setw(WIDTH / 2) << "同书索引: " << index\_ << std::endl;

std::cout << std::setw(WIDTH / 2) << "书名: " << name\_ << std::endl;

std::cout << std::setw(WIDTH / 2) << "作者: " << author\_ << std::endl;

std::cout << std::setw(WIDTH / 2) << "出版信息: " << publish\_ << std::endl;

HighlightPrint((on\_shelf\_ ? "在架上\n" : "已借出\n" ));

return;

}

void Book::update()

{

Remove(all\_books, isbn\_, index\_);

all\_books.push\_back(\*this);

WriteBooks();

ReadAll();

return;

}

void Book::wanted(int reader\_id)

{

Reader reader = readers[Find(readers, reader\_id)];

std::vector<Info> tmp = same\_state(WANTED);

time\_t now = time(NULL);

time\_t start\_time = now;

time\_t end\_time = now + days\_ \* reader.credit() \* SECONDS\_A\_DAY / CREDIT;

if (tmp.size() != 0) // 有人预约

{

if (tmp[0].end\_time < now)

{

Remove(info\_, tmp[0].reader);

Remove(info\_, reader\_id);

info\_.push\_back(Info(reader\_id, HOLDING, start\_time, end\_time));

on\_shelf\_ = false;

reader.del\_book(\*this);

reader.add\_book(\*this);

reader.update();

update();

HighlightPrint("借阅成功! \n");

info(reader\_id).print\_duration();

return;

}

if (tmp[0].reader == reader\_id)

{

if (same\_state(HOLDING).size() == 0)

{

change\_state(reader\_id, HOLDING);

info(reader\_id).start\_time = start\_time;

info(reader\_id).end\_time = end\_time;

on\_shelf\_ = false;

reader.del\_book(\*this);

reader.add\_book(\*this);

reader.update();

update();

HighlightPrint("您已提前预约, 借阅成功!\n");

info(reader\_id).print\_duration();

return;

}

HighlightPrint("您已预约过, 不能重复预约!\n");

return;

}

HighlightPrint("预约失败, 已有人预约!\n");

return;

}

tmp = same\_state(OVERDUE);

if (tmp.size() != 0)

{

if (tmp[0].reader == reader\_id)

{

HighlightPrint("请尽快归还本书!\n");

return;

}

Remove(info\_, reader\_id);

info\_.push\_back(Info(reader\_id, REQUEST, start\_time, start\_time + 3 \* SECONDS\_A\_DAY));

reader.del\_book(\*this);

reader.add\_book(\*this);

reader.update();

update();

HighlightPrint("预约成功, 由于上一位读者现逾期未还, 您从即日起拥有三天的此书优先借阅权\n");

info(reader\_id).print\_duration();

return;

}

tmp = same\_state(HOLDING);

if (tmp.size() != 0)

{

if (tmp[0].reader == reader\_id)

{

HighlightPrint("您正在借阅此书!\n");

return;

}

info(tmp[0].reader).state = WANTED;

info(tmp[0].reader).end\_time = (now + info(tmp[0].reader).end\_time) / 2;

Remove(info\_, reader\_id);

info\_.push\_back(Info(reader\_id, REQUEST, start\_time, tmp[0].end\_time + 3 \* SECONDS\_A\_DAY));

reader.del\_book(\*this);

reader.add\_book(\*this);

reader.update();

update();

HighlightPrint("预约成功, 您从上位读者还书日起拥有三天的此书优先借阅权\n");

info(reader\_id).print\_duration();

return;

}

Remove(info\_, reader\_id);

info\_.push\_back(Info(reader\_id, HOLDING, start\_time, end\_time));

on\_shelf\_ = false;

reader.del\_book(\*this);

reader.add\_book(\*this);

reader.update();

update();

HighlightPrint("借阅成功! \n");

info(reader\_id).print\_duration();

return;

}

// book.h

#ifndef BOOK\_H

#define BOOK\_H

#include <string>

#include <vector>

#include <ctime>

enum State

{

HOLDING, // 正常持有

REQUEST, // 预约此书

WANTED, // 被其他读者催还

OVERDUE, // 逾期未还

RETURNED // 已还

};

const int DAYS = 30;

struct Info

{

Info() {}

Info(int reader, int state, time\_t start\_time, time\_t end\_time): reader(reader), state(state), start\_time(start\_time), end\_time(end\_time) {}

bool operator ==(const int &reader\_id) { return (reader == reader\_id); }

friend std::ifstream &operator >>(std::ifstream &in, Info &info);

friend std::ofstream &operator <<(std::ofstream &out, const Info &info);

void print();

void print\_duration();

int reader; // 读者ID

int state; // 借阅状态

time\_t start\_time; // 开始日期

time\_t end\_time; // 结束日期

};

class Book

{

public:

Book() {}

Book(std::string isbn, std::string name, std::string author, std::string publish, int days = DAYS);

~Book() {}

bool operator ==(const Book &b) { return ((isbn\_ == b.isbn\_) && (index\_ == b.index\_)); }

friend std::ifstream &operator >>(std::ifstream &in, Book &book);

friend std::ofstream &operator <<(std::ofstream &out, const Book &book);

bool on\_shelf() { return on\_shelf\_; }

Info &info(int reader\_id); // 返回该读者借阅此书的信息

int days() { return days\_; }

int index() { return index\_; }

std::string author() { return author\_; }

std::string isbn() { return isbn\_; }

std::string name() { return name\_; }

std::string publish() { return publish\_; }

std::vector<Info> info() { return info\_; }

std::vector<Info> same\_state(int state);

void change\_state() { on\_shelf\_ = !on\_shelf\_; } // 借出/归还

void change\_state(int reader\_id, int state);

void display\_now\_state();

void give\_back(int reader\_id); // 读者归还

void print();

void update();

void wanted(int reader\_id); // 有读者需要, 若在架上则成功借出, 否则变更相关状态.

private:

std::string isbn\_; // ISBN号, 10位数字, unsigned不能完全存下, 故用string.

std::string name\_; // 书名

std::string author\_; // 作者

std::string publish\_; // 出版信息

int index\_; // 同一版书籍可能有多本, ISBN相同的书籍利用index相互区分.

bool on\_shelf\_; // 当前是否在架上

int days\_; // 正常情况下可借阅的天数

std::vector<Info> info\_;// 与此书相关的所有读者信息

};

#endif

// file.h

#ifndef FILE\_H

#define FILE\_H

#include <iostream>

#include <fstream>

void ReadAdmins();

void ReadAll(); // 读取所有文件中的信息

void ReadBooks();

void ReadReaders();

void ReadUsers();

void WriteAdmins();

void WriteAll(); // 将所有全局信息写入文件

void WriteBooks();

void WriteReaders();

#endif

// file.cpp

#include "file.h"

#include "user.h"

#include "global.h"

#include <vector>

void ReadAdmins()

{

std::ifstream in("./data/admins.txt", std::ios::in);

admins.clear();

Administrator tmp;

while (in >> tmp)

{

admins.push\_back(tmp);

tmp = Administrator();

}

}

void ReadBooks()

{

std::ifstream in("./data/books.txt", std::ios::in);

all\_books.clear();

Book tmp;

while (in >> tmp)

{

all\_books.push\_back(tmp);

tmp = Book();

}

}

void ReadReaders()

{

std::ifstream in("./data/readers.txt", std::ios::in);

readers.clear();

Reader tmp;

while (in >> tmp)

{

readers.push\_back(tmp);

tmp = Reader();

}

}

void ReadUsers()

{

users.clear();

User\* tmp;

for (std::vector<Administrator>::iterator it = admins.begin(); it != admins.end(); ++it)

{

tmp = &\*it;

users.push\_back(tmp);

}

for (std::vector<Reader>::iterator it = readers.begin(); it != readers.end(); ++it)

{

tmp = &\*it;

users.push\_back(tmp);

}

}

void ReadAll()

{

ReadAdmins();

ReadReaders();

ReadUsers();

ReadBooks();

}

void WriteAdmins()

{

std::ofstream out("./data/admins.txt", std::ios::out|std::ios::trunc);

for (std::vector<Administrator>::iterator it = admins.begin(); it != admins.end(); ++it)

out << \*it;

}

void WriteBooks()

{

std::ofstream out("./data/books.txt", std::ios::out|std::ios::trunc);

for (std::vector<Book>::iterator it = all\_books.begin(); it != all\_books.end(); ++it)

out << \*it;

}

void WriteReaders()

{

std::ofstream out("./data/readers.txt", std::ios::out|std::ios::trunc);

for (std::vector<Reader>::iterator it = readers.begin(); it != readers.end(); ++it)

out << \*it;

}

void WriteAll()

{

WriteAdmins();

WriteReaders();

WriteBooks();

}

// global.h

#ifndef GLOBAL\_H

#define GLOBAL\_H

#include "user.h"

#include <vector>

#include <iomanip>

#include <iostream>

#include <cmath>

#include <thread>

#include <chrono>

#include <utility>

extern std::vector<User\*> users;

extern std::vector<Administrator> admins;

extern std::vector<Reader> readers;

extern std::vector<Book> all\_books; // 与Reader::books\_区分

typedef std::pair<int, char> Key; // first = times, second = ch;

const Key ENTER = std::make\_pair(1, char(13));

const Key ESCAPE = std::make\_pair(1, char(27));

const Key UP = std::make\_pair(2, char(72));

const Key DOWN = std::make\_pair(2, char(80));

const Key LEFT = std::make\_pair(2, char(75));

const Key RIGHT = std::make\_pair(2, char(77));

const int SECONDS\_A\_DAY = 24 \* 3600;

Key MyGetCh();

bool CompareHot(Book b1, Book b2);

bool HighlightPrint(int setw, std::string text, int color = 9);

bool HighlightPrint(std::string text, int color = 9);

bool ValidPassword(std::string pwd); // 检查是否符合6~15位数字字母密码标准

int Find(std::vector<Book> v, std::string isbn, int index);

int Find(std::vector<User\*> v, int id); // 返回vector中找到的索引, 未找到则返回-1

std::string Decode(int id, std::string cipher); // 解密

std::string GetPass(); // 输入密码

std::string RandomPass(int digits = 6); // 返回digits位随机密码

std::vector<Book> Find(std::vector<Book> v, std::string isbn);

std::vector<Book> HotBook(int num = 3); // 借阅排行榜

Token Login(); // 登录, 返回令牌

void ClearScreen(); // 保留标题的清屏函数

void EBook(); // 电子资源界面

void Exit(); // 退出系统

void MediatePrint(std::string text); // 居中打印字符串

void PrintBooks(std::vector<Book> v);

void PrintBooksOfReader(std::vector<Book> v, int reader\_id);

void Remove(std::vector<Book> &v, std::string isbn, int index);

void Remove(std::vector<Book> &v, std::vector<Book> dels);

void Scroll(std::string text); // 滚动播放文本

void Title(); // 系统标题

void Wait(int milliseconds = 1000); // 等待毫秒数

void Welcome(); // 欢迎界面

template <class T>

std::string Encode(T user) // 加密

{

int id = user.id\_;

std::string clear = user.password\_;

std::vector<int> shifts;

std::string cipher;

for (int i = 0; id >= 10; ++i)

{

shifts.push\_back(((i % 2) ? -1 : 1) \* (id % 10));

id /= 10;

}

int i = 0;

while (clear.size())

{

cipher.push\_back(clear.back() + shifts[i]);

clear.pop\_back();

if (++i > shifts.size() - 1)

i = 0;

}

return cipher;

}

template <class A, class B>

int Find(std::vector<A> v, B id)

{

for (int i = 0; i < v.size(); ++i)

{

if (v[i] == id)

return i;

}

return -1;

}

template <class A, class B>

void Remove(std::vector<A> &v, B id)

{

for (std::vector<A>::iterator it = v.begin(); it != v.end(); ++it)

{

if (\*it == id)

{

v.erase(it);

break;

}

}

}

#endif

// global.cpp

#include "global.h"

#include <stdlib.h>

#include <windows.h>

#include <algorithm>

#include <iomanip>

#include <ctime>

#include <conio.h>

#include "token.h"

std::vector<User\*> users;

std::vector<Administrator> admins;

std::vector<Reader> readers;

std::vector<Book> all\_books;

Key MyGetCh()

{

int ch = getch();

if (kbhit())

return std::make\_pair(2, getch());

return std::make\_pair(1, ch);

}

bool CompareHot(Book b1, Book b2)

{

return (b1.info().size() > b2.info().size());

}

bool HighlightPrint(int setw, std::string text, int color)

{

HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (handle == 0)

return false;

BOOL ret = SetConsoleTextAttribute(handle, WORD(color));

std::cout << std::setw(setw) << text;

ret = SetConsoleTextAttribute(handle, WORD(7));

return (ret == true);

}

bool HighlightPrint(std::string text, int color)

{

return HighlightPrint((WIDTH + text.size()) / 2, text, color);

}

bool ValidPassword(std::string pwd)

{

return ((pwd.size() >= 6) && (pwd.size() <= 15));

}

int Find(std::vector<Book> v, std::string isbn, int index)

{

for (int i = 0; i < v.size(); ++i)

{

if ((v[i].isbn() == isbn) && (v[i].index() == index))

return i;

}

return -1;

}

int Find(std::vector<User\*> v, int id)

{

for (int i = 0; i < v.size(); ++i)

{

if (v[i]->id() == id)

return i;

}

return -1;

}

std::string Decode(int id, std::string cipher)

{

std::vector<int> shifts;

for (int i = 0; id >= 10; ++i)

{

shifts.push\_back(((i % 2) ? 1 : -1) \* (id % 10));

id /= 10;

}

std::string clear;

int i = 0;

while (cipher.size())

{

clear.insert(clear.begin(), cipher.front() + shifts[i]);

cipher.erase(cipher.begin());

if (++i > shifts.size() - 1)

i = 0;

}

return clear;

}

std::string GetPass()

{

int c;

std::string password;

std::string::iterator it = password.end();

while ((c = getch()) != '\r')

{

switch (c)

{

case 0:

{

getch();

std::cout << '\a';

break;

}

case 224:

{

switch (getch())

{

case 75: // left

{

if (it != password.begin())

{

--it;

std::cout << '\b';

}

else

std::cout << '\a';

break;

}

case 77: // right

{

if (it != password.end())

{

++it;

std::cout << '\*';

}

else

std::cout << '\a';

break;

}

default:

{

std::cout << '\a';

break;

}

}

break;

}

case 163:

{

getch();

std::cout << '\a';

break;

}

case '\b':

{

if (password.size() != 0 && it != password.begin())

{

std::cout << '\b';

for (int i = 0; i < password.end() - it; ++i)

std::cout << '\*';

std::cout << ' ';

for (int i = 0; i < password.end() - it + 1; ++i)

std::cout << '\b';

password.erase(it - 1);

--it;

}

else

std::cout << '\a';

break;

}

default:

{

if (isalnum(c) || ispunct(c))

{

password.insert(it, c);

++it;

for (int i = 0; i < password.end() - it + 1; ++i)

std::cout << '\*';

for (int i = 0; i < password.end() - it; ++i)

std::cout << '\b';

}

else

std::cout << '\a';

break;

}

}

}

std::cout << std::endl;

return password;

}

std::string RandomPass(int digits)

{

char set[] = {'0','1','2','3','4','5','6','7','8','9',

'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',

'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

std::string pwd;

for (int i = 0; i < digits; ++i)

{

srand((unsigned int)((i + 1) \* time(NULL)));

pwd.push\_back(set[rand() % 52]);

}

return pwd;

}

std::vector<Book> Find(std::vector<Book> v, std::string isbn)

{

std::vector<Book> rst;

for (std::vector<Book>::iterator it = v.begin(); it != v.end(); ++it)

{

if (it->isbn() == isbn)

rst.push\_back(\*it);

}

return rst;

}

std::vector<Book> HotBook(int num)

{

std::sort(all\_books.begin(), all\_books.end(), CompareHot);

num = (num <= all\_books.size() ? num : all\_books.size());

return std::vector<Book>(all\_books.begin(), all\_books.begin() + num);

}

Token Login()

{

MediatePrint("请登录...\n");

int id;

std::string pwd;

std::cout << std::setw(WIDTH / 2) << "请输入您的学号(工号): ";

if (!(std::cin >> id))

{

std::cin.clear();

std::cin.sync();

return Token(-1, "");

}

std::cout << std::setw(WIDTH / 2) << "请输入您的密码: ";

pwd = GetPass();

return Token(id, pwd);

}

void ClearScreen()

{

system("cls");

Title();

}

void EBook()

{

HighlightPrint("请登录 http://lib.tsinghua.edu.cn 访问查阅电子资源库!");

MyGetCh();

}

void Exit()

{

ClearScreen();

MediatePrint("确认退出? [y/n] \n");

Key ch = MyGetCh();

if (ch == std::make\_pair(1, 'y') || ch == std::make\_pair(1, 'Y'))

{

MediatePrint("再见! \n");

exit(1);

}

}

void MediatePrint(std::string text)

{

std::cout << std::setw((WIDTH + text.size()) / 2) << text;

}

void PrintBooks(std::vector<Book> v)

{

ClearScreen();

std::string seperator(80, '-');

for (std::vector<Book>::iterator it = v.begin(); it != v.end(); ++it)

{

it->print();

std::cout << seperator << std::endl;

}

return;

}

void PrintBooksOfReader(std::vector<Book> v, int reader\_id)

{

std::string seperator(80, '-');

for (std::vector<Book>::iterator it = v.begin(); it != v.end(); ++it)

{

it->print();

it->info(reader\_id).print();

std::cout << seperator << std::endl;

}

return;

}

void Remove(std::vector<Book> &v, std::string isbn, int index)

{

for (std::vector<Book>::iterator it = v.begin(); it != v.end(); ++it)

{

if ((it->isbn() == isbn) && (it->index() == index))

{

v.erase(it);

break;

}

}

}

void Remove(std::vector<Book> &v, std::vector<Book> dels)

{

for (std::vector<Book>::iterator it = dels.begin(); it != dels.end(); ++it)

Remove(v, \*it);

}

void Title()

{

HighlightPrint("图书管理系统 v1.60.1\n\n");

}

void Wait(int milliseconds)

{

std::this\_thread::sleep\_for(std::chrono::milliseconds(milliseconds));

}

// menu.h

#ifndef MENU\_H

#define MENU\_H

#include <string>

#include <vector>

#include <utility>

struct MenuNode;

typedef std::pair<std::string, MenuNode\*> Option;

// 每个选项由名称和目标菜单构成.

struct MenuNode

{

std::vector<Option> options;

MenuNode\* previous;

void append(Option option);

void show(int index); // 选中的选项索引为index

};

#endif

// menu.cpp

#include "menu.h"

#include "global.h"

void MenuNode::append(Option option)

{

options.push\_back(option);

}

void MenuNode::show(int index)

{

ClearScreen();

for (int i = 0; i < options.size(); ++i)

{

if (i == index)

{

HighlightPrint(options[i].first + '\n');

continue;

}

MediatePrint(options[i].first + '\n');

}

}

// token.h

#ifndef TOKEN\_H

#define TOKEN\_H

#include <string>

#include <vector>

class Token

{

public:

Token() {}

Token(int id, std::string password): id\_(id), password\_(password) {}

~Token() {}

int id() { return id\_; }

int auth(); // 返回认证身份, 认证失败返回-1.

private:

int id\_;

std::string password\_;

};

#endif

// token.cpp

#include "token.h"

#include "global.h"

int Token::auth()

{

int index = Find(users, id\_);

if (index < 0)

return -1;

User\* tmp = users[Find(users, id\_)];

if (password\_ == tmp->password\_)

return tmp->identity\_;

return -1;

}

// user.h

#ifndef USER\_H

#define USER\_H

#include <string>

#include <iostream>

#include <iomanip>

#include <vector>

#include "book.h"

const int WIDTH = 80;

enum UserType { ADMINISTRATOR, READER };

const int CREDIT = 80; // 初始信用积分

class User

{

public:

User() {}

User(int identity, int id, std::string password): identity\_(identity), id\_(id), password\_(password) {}

virtual ~User() {}

friend class Token;

template <class T>

friend std::string Encode(T user);

bool operator ==(const int &id) { return id\_ == id; }

virtual void print() = 0; // 显示用户基本信息

virtual void update() = 0; // 更新全局vector及文件

int id() { return id\_; }

int identity() {return identity\_; }

void set\_password(); // 重设密码

protected:

int identity\_; // 用户身份

int id\_; // 用户ID

std::string password\_; // 密码

};

class Administrator: public User

{

public:

Administrator() {}

Administrator(int id, std::string password): User(ADMINISTRATOR, id, password) {}

virtual ~Administrator() {}

friend std::ifstream &operator >>(std::ifstream &in, Administrator &admin);

friend std::ofstream &operator <<(std::ofstream &out, const Administrator &admin);

void add\_book(); // 新书上架

void add\_user(); // 增加新用户

void all\_book(); // 查看所有书籍信息

void all\_user(); // 查看所有用户信息

void del\_book(); // 旧书下架

void del\_user(); // 删除用户

void print() { std::cout << std::setw(WIDTH / 2) << "管理员: " << id\_ << std::endl; }

void update();

private:

};

class Reader: public User

{

public:

Reader() { books\_.reserve(256); }

Reader(int id, std::string password, std::string name): User(READER, id, password), name\_(name), credit\_(CREDIT) { books\_.reserve(256); }

virtual ~Reader() {}

friend std::ifstream &operator >>(std::ifstream &in, Reader &reader);

friend std::ofstream &operator <<(std::ofstream &out, const Reader &reader);

bool change\_state(Book book, int state); // 变更book的状态为state

int change\_credit(int diff) { return credit\_ += diff; } // 信用积分变化

int credit() { return credit\_; }

std::string name() { return name\_; }

std::vector<Book> books(int state = 0);

void add\_book(Book book);

void del\_book(Book book);

void history(); // 借阅历史界面

void print() { std::cout << std::setw(WIDTH / 2 - 10) << "读者: " << id\_ << " " << name\_ << " 当前信用积分: " << credit\_ << std::endl; }

void recommended(); // 馆长推荐界面

void give\_back(); // 读者还书界面

void search(); // 搜索书籍界面

void update();

private:

std::string name\_; // 读者姓名

int credit\_; // 信用积分

std::vector<Book> books\_; // 与该读者相关的书(包括借阅过, 持有, 预约等)

};

#endif

// user.cpp

#include "user.h"

#include "global.h"

#include <conio.h>

#include "file.h"

#include <cmath>

#include "menu.h"

void User::set\_password()

{

ClearScreen();

std::cout << std::setw(WIDTH / 2) << "请输入密码: ";

std::string origin = GetPass();

if (origin != password\_)

{

HighlightPrint("密码错误!");

MyGetCh();

return;

}

std::cout << std::setw(WIDTH / 2) << "新密码: ";

std::string new\_pwd = GetPass();

if (!ValidPassword(new\_pwd))

{

HighlightPrint("请设置6~15位密码!");

MyGetCh();

return;

}

std::cout << std::setw(WIDTH / 2) << "重复新密码: ";

std::string confirm = GetPass();

if (new\_pwd != confirm)

{

HighlightPrint("两次密码不一致!");

MyGetCh();

return;

}

password\_ = new\_pwd;

update();

HighlightPrint("设置成功!\n");

MyGetCh();

return;

}

std::ifstream &operator >>(std::ifstream &in, Administrator &admin)

{

in >> admin.identity\_;

in >> admin.id\_;

std::string cipher;

in >> cipher;

admin.password\_ = Decode(admin.id\_, cipher);

return in;

}

std::ofstream &operator <<(std::ofstream &out, const Administrator &admin)

{

out << admin.identity\_ << '\n';

out << admin.id\_ << '\n';

out << Encode(admin) << '\n';

return out;

}

void Administrator::add\_user()

{

ClearScreen();

std::vector<std::string> identities;

identities.push\_back("管理员");

identities.push\_back("读者");

int k = 0;

while (true)

{

ClearScreen();

MediatePrint("请选择新用户的身份: \n");

for (int i = 0; i < 2; ++i)

{

if (i == k)

{

HighlightPrint(identities[i] + '\n');

continue;

}

MediatePrint(identities[i] + '\n');

}

Key ch = MyGetCh();

if (ch == RIGHT || ch == ENTER)

{

ClearScreen();

HighlightPrint(identities[k] + '\n');

break;

}

if (ch == LEFT || ch == ESCAPE)

return;

if (ch == UP || ch == DOWN)

k = !k;

}

int new\_id = 0;

for (std::vector<User\*>::iterator it = users.begin(); it != users.end(); ++it)

{

if (new\_id <= (\*it)->id())

new\_id = (\*it)->id() + 1;

}

std::cout << std::setw(WIDTH / 2) << "ID: " << new\_id << std::endl;

std::string new\_pwd = RandomPass();

MediatePrint("正在打印密码单...请稍候...\n\n");

std::string space(10, ' ');

std::cout << space << '|' << std::string(58,'-') << '|' << std::endl;

Wait();

std::cout << space << '|' << std::setw(59) << '|' << std::endl;

Wait();

std::cout << space << '|' << space + space << "ID: " << new\_id << std::setw(34 - int(log10(new\_id))) << '|' << std::endl;

Wait();

std::cout << space << '|' << space + space << "密码: " << new\_pwd << std::setw(27) << '|' << std::endl;

Wait();

std::cout << space << '|' << std::setw(59) << '|' << std::endl;

Wait();

std::cout << space << '|' << std::string(58,'-') << '|' << std::endl << std::endl;

// 密码单打印完毕.

MediatePrint("密码单打印完毕! \n");

MediatePrint("请尽快修改密码! \n");

if (!k) // 管理员

{

Administrator new\_admin(new\_id, new\_pwd);

new\_admin.update();

}

else // 读者

{

std::cout << std::setw(WIDTH / 2) << "姓名: ";

std::string new\_name;

std::cin >> new\_name;

Reader new\_reader(new\_id, new\_pwd, new\_name);

new\_reader.update();

}

MyGetCh();

return;

}

void Administrator::del\_user()

{

ClearScreen();

std::cout << std::setw(WIDTH / 2) << "学号(工号): ";

int del\_id;

if (!(std::cin >> del\_id))

{

HighlightPrint("输入错误! \n");

MyGetCh();

return;

}

int index = Find(users, del\_id);

if (index < 0)

{

HighlightPrint("用户不存在! \n");

MyGetCh();

return;

}

if (!(users[index]->identity())) // 管理员

{

HighlightPrint("无权进行此操作! \n");

MyGetCh();

return;

}

else

{

users[index]->print();

Reader reader = readers[Find(readers, del\_id)];

if (reader.books(HOLDING).size() != 0 || reader.books(WANTED).size() != 0 || reader.books(OVERDUE).size() != 0)

{

HighlightPrint("该用户还有书未还! 删除失败!\n");

MyGetCh();

return;

}

MediatePrint("确认删除? [y/n] \n");

Key ch = MyGetCh();

if (ch == std::make\_pair(1, 'y') || ch == std::make\_pair(1, 'Y'))

{

Remove(readers, del\_id);

WriteReaders();

ReadAll();

MediatePrint("用户删除成功!\n");

MyGetCh();

return;

}

MediatePrint("用户未删除!\n");

MyGetCh();

return;

}

}

void Administrator::all\_book()

{

PrintBooks(all\_books);

MyGetCh();

return;

}

void Administrator::add\_book()

{

ClearScreen();

std::string new\_isbn, new\_name, new\_author, new\_publish;

std::cout << std::setw(WIDTH / 2) << "ISBN: ";

std::cin >> new\_isbn;

std::cin.get(); // 读掉回车

if (new\_isbn.size() != 10 && new\_isbn.size() != 13)

{

HighlightPrint("输入错误!\n");

MyGetCh();

return;

}

for (std::string::iterator it = new\_isbn.begin(); it != new\_isbn.end(); ++it)

{

if (!isdigit(\*it))

{

HighlightPrint("输入错误!\n");

MyGetCh();

return;

}

}

std::cout << std::setw(WIDTH / 2) << "书名: ";

getline(std::cin, new\_name);

std::cout << std::setw(WIDTH / 2) << "作者: ";

getline(std::cin, new\_author);

std::cout << std::setw(WIDTH / 2) << "出版信息: ";

getline(std::cin, new\_publish);

Book new\_book(new\_isbn, new\_name, new\_author, new\_publish);

new\_book.update();

MediatePrint("添加成功!\n");

MyGetCh();

return;

}

void Administrator::del\_book()

{

ClearScreen();

std::string del\_isbn;

std::cout << std::setw(WIDTH / 2) << "ISBN: ";

std::cin >> del\_isbn;

std::vector<Book> rst = Find(all\_books, del\_isbn);

if (rst.size() == 0)

{

HighlightPrint("未找到该书!\n");

MyGetCh();

return;

}

PrintBooks(rst);

MediatePrint("确认下架? [y全部下架; 或输入相应索引号] \n");

Key ch = MyGetCh();

if (ch == std::make\_pair(1, 'y') || ch == std::make\_pair(1, 'Y'))

{

for (std::vector<Book>::iterator it = rst.begin(); it != rst.end(); ++it)

{

std::vector<Info> tmp = it->info();

for (std::vector<Info>::iterator iter = tmp.begin(); iter != tmp.end(); ++iter)

{

if (iter->state == HOLDING || iter->state == WANTED || iter->state == OVERDUE)

{

HighlightPrint("该书仍未被归还! 删除失败!\n");

MyGetCh();

return;

}

}

}

Remove(all\_books, rst);

WriteBooks();

ReadAll();

MediatePrint("删除成功!\n");

MyGetCh();

return;

}

int del\_index, index;

if (!(std::cin >> del\_index))

{

HighlightPrint("输入错误!\n");

MyGetCh();

return;

}

if ((index = Find(all\_books, del\_isbn, del\_index)) < 0)

{

HighlightPrint("输入错误!\n");

MyGetCh();

return;

}

Book tmp = all\_books[Find(all\_books, del\_isbn, del\_index)];

std::vector<Info> info = tmp.info();

for (std::vector<Info>::iterator it = info.begin(); it != info.end(); ++it)

{

if (it->state == HOLDING || it->state == WANTED || it->state == OVERDUE)

{

HighlightPrint("该书仍未被归还! 删除失败!\n");

MyGetCh();

return;

}

}

Remove(all\_books, del\_isbn, del\_index);

WriteBooks();

ReadAll();

MediatePrint("删除成功!\n");

MyGetCh();

return;

}

void Administrator::all\_user()

{

ClearScreen();

for (std::vector<User\*>::iterator it = users.begin(); it != users.end(); ++it)

(\*it)->print();

MyGetCh();

return;

}

void Administrator::update()

{

Remove(admins, id\_);

admins.push\_back(\*this);

WriteAdmins();

ReadAll();

return;

}

std::ifstream &operator >>(std::ifstream &in, Reader &reader)

{

in >> reader.identity\_;

in >> reader.id\_;

std::string cipher;

in >> cipher;

reader.password\_ = Decode(reader.id\_, cipher);

in >> reader.name\_;

in >> reader.credit\_;

char start\_flag;

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

Book tmp;

in >> tmp;

reader.books\_.push\_back(tmp);

}

}

}

else

in.seekg(-1, std::ios::cur);

return in;

}

std::ofstream &operator <<(std::ofstream &out, const Reader &reader)

{

out << reader.identity\_ << '\n';

out << reader.id\_ << '\n';

out << Encode(reader) << '\n';

out << reader.name\_ << '\n';

out << reader.credit\_ << '\n';

out << "\*\n";

for (std::vector<Book>::const\_iterator it = reader.books\_.begin(); it != reader.books\_.end(); ++it)

out << \*it;

out << "#\n\n";

return out;

}

bool Reader::change\_state(Book book, int state)

{

for (std::vector<Book>::iterator it = books\_.begin(); it != books\_.end(); ++it)

{

if (\*it == book)

{

it->change\_state(id\_, state);

update();

return true;

}

}

return false;

}

void Reader::add\_book(Book book)

{

books\_.push\_back(book);

}

void Reader::del\_book(Book book)

{

Remove(books\_, book);

}

std::vector<Book> Reader::books(int state)

{

std::vector<Book> rst;

for (std::vector<Book>::iterator it = books\_.begin(); it != books\_.end(); ++it)

{

if (it->info(id\_).state == state)

rst.push\_back(\*it);

}

return rst;

}

void Reader::history()

{

ClearScreen();

PrintBooksOfReader(books(RETURNED), id\_);

MyGetCh();

return;

}

void Reader::recommended()

{

ClearScreen();

std::vector<Book> recommended = HotBook(5);

if (recommended.size() == 0)

{

MediatePrint("馆长还没有推荐什么好书哦!\n");

MyGetCh();

return;

}

std::vector<std::string> book\_names;

for (std::vector<Book>::iterator it = recommended.begin(); it != recommended.end(); ++it)

{

book\_names.push\_back(it->name());

}

int k = 0;

while (true)

{

ClearScreen();

for (int i = 0; i < book\_names.size(); ++i)

{

if (i == k)

{

HighlightPrint(book\_names[i] + '\n');

continue;

}

MediatePrint(book\_names[i] + '\n');

}

Key ch = MyGetCh();

if (ch == RIGHT || ch == ENTER)

{

ClearScreen();

Book target = recommended[k];

target.print();

target.display\_now\_state();

MediatePrint("借阅/预约此书? [y/n] \n");

Key tmp = MyGetCh();

if (tmp == std::make\_pair(1, 'y') || tmp == std::make\_pair(1, 'Y'))

{

target.wanted(id\_);

}

MyGetCh();

return;

}

if (ch == LEFT || ch == ESCAPE)

return;

if (ch == UP)

{

if (--k < 0)

k = book\_names.size() - 1;

continue;

}

if (ch == DOWN)

{

if (++k > book\_names.size() - 1)

k = 0;

continue;

}

}

MyGetCh();

return;

}

void Reader::give\_back()

{

ClearScreen();

std::vector<Book> can\_return;

std::vector<Book> tmp = books(HOLDING);

can\_return.insert(can\_return.end(), tmp.begin(), tmp.end());

tmp = books(WANTED);

can\_return.insert(can\_return.end(), tmp.begin(), tmp.end());

tmp = books(OVERDUE);

can\_return.insert(can\_return.end(), tmp.begin(), tmp.end());

if (can\_return.size() == 0)

{

MediatePrint("您没有需要还的书!\n");

MyGetCh();

return;

}

std::vector<std::string> book\_names;

for (std::vector<Book>::iterator it = can\_return.begin(); it != can\_return.end(); ++it)

{

book\_names.push\_back(it->name());

}

int k = 0;

while (true)

{

ClearScreen();

for (int i = 0; i < book\_names.size(); ++i)

{

if (i == k)

{

HighlightPrint(book\_names[i] + '\n');

continue;

}

MediatePrint(book\_names[i] + '\n');

}

Key ch = MyGetCh();

if (ch == RIGHT || ch == ENTER)

{

ClearScreen();

Book target = can\_return[k];

target.print();

target.display\_now\_state();

MediatePrint("归还此书? [y/n] \n");

Key tmp = MyGetCh();

if (tmp == std::make\_pair(1, 'y') || tmp == std::make\_pair(1, 'Y'))

{

target.give\_back(id\_);

}

MyGetCh();

return;

}

if (ch == LEFT || ch == ESCAPE)

return;

if (ch == UP)

{

if (--k < 0)

k = book\_names.size() - 1;

continue;

}

if (ch == DOWN)

{

if (++k > book\_names.size() - 1)

k = 0;

continue;

}

}

MyGetCh();

return;

}

void Reader::search()

{

ClearScreen();

std::cout << std::setw(WIDTH / 2) << "关键词/ISBN: ";

std::vector<std::string> keywords;

std::string tmp;

while (std::cin >> tmp)

{

keywords.push\_back(tmp);

if (std::cin.get() == '\n')

break;

}

std::vector<Book> search\_result;

for (std::vector<Book>::iterator it = all\_books.begin(); it != all\_books.end(); ++it)

{

for (std::vector<std::string>::iterator iter = keywords.begin(); iter != keywords.end(); ++iter)

{

if (it->name().find(\*iter) != std::string::npos || it->isbn().find(\*iter) != std::string::npos)

search\_result.push\_back(\*it);

}

}

std::vector<std::string> book\_names;

for (std::vector<Book>::iterator it = search\_result.begin(); it != search\_result.end(); ++it)

{

book\_names.push\_back(it->name());

}

int k = 0;

while (true)

{

ClearScreen();

for (int i = 0; i < book\_names.size(); ++i)

{

if (i == k)

{

HighlightPrint(book\_names[i] + '\n');

continue;

}

MediatePrint(book\_names[i] + '\n');

}

Key ch = MyGetCh();

if (ch == RIGHT || ch == ENTER)

{

ClearScreen();

Book target = search\_result[k];

target.print();

target.display\_now\_state();

MediatePrint("借阅/预约此书? [y/n] \n");

Key tmp = MyGetCh();

if (tmp == std::make\_pair(1, 'y') || tmp == std::make\_pair(1, 'Y'))

{

target.wanted(id\_);

}

MyGetCh();

return;

}

if (ch == LEFT || ch == ESCAPE)

return;

if (ch == UP)

{

if (--k < 0)

k = book\_names.size() - 1;

continue;

}

if (ch == DOWN)

{

if (++k > book\_names.size() - 1)

k = 0;

continue;

}

}

MyGetCh();

return;

}

void Reader::update()

{

Remove(readers, id\_);

readers.push\_back(\*this);

WriteReaders();

ReadAll();

return;

}

**附录1：评分表**

课程名称： 计算机程序设计基础2

|  |  |  |
| --- | --- | --- |
| **项 目** | **评 价** | |
| 设计方案的合理性与创新性 | **3×2** |  |
| 设计与调试结果 | **4×2** |  |
| 设计说明书的质量 | **1×2** |  |
| 程序基本要求涵盖情况 | **4×2** |  |
| 程序代码编写素养情况 | **2×2** |  |
| 课程设计周表现情况 | **1×2** |  |
| 综合成绩 | **15×2** |  |

教师签名：

日 期：