**OOPDS HW2**

**學號：113550021 姓名：陳孟楷**

**Overview**

* **TIPs**
* **Data Structure**
* **Operations**
* **Main Functions**
* **Optimal Enhancement**

**TIPs**

1. **加粗藍字**是通常代表的是程式碼內的內容
2. 為了編排流暢，內文中也有optimal enhancement 的部份，**(OE)** 指的是用於optimal enhancement 的部分

**Data Structure**

* **Book**: store the information of a book, including the **title**, the published **year**, the **author** and the **number** of copies

class Book

{

public:

string title;

int year;

string author;

int number;

Book(string t, int y, string a, int n)

: title(t), year(y), author(a), number(n){}

};

* **Record (OE)**: store the records an user or an admin has done, including the **operation**, the **title** and the **author** (operation only contains “**add**” a book,” **check out**” a book and ”**return**” a book)

class Record

{

public:

string operation, title, author;

int year;

Record(string o, string t, int y, string a): operation(o), title(t), year(y), author(a) {}

};

* **User (OE)**: store the information of an user, including the **username** and the **password**

class User

{

public:

string username, password;

vector<Record> record;

User(string u, string p): username(u), password(p){}

};

* **Admin (OE)**: store the information of an admin, including the **adminname** and **password**

class Admin

{

public:

string adminname, password;

vector<Record> record;

Admin(string a, string p): adminname(a), password(p){}

};

* **Library**: used to control the whole library management system

class Library {...}; // 以下省略

* Inside **Library – private**, you would see the data structures below:
  + **books**: store all the books
  + **users**: store all the users
  + **admins**: store all the admins
  + **current\_user**: indicating the current user. If none, it would be **nullptr**
  + **current\_admin**: indicating the current admin. If none, it would be **nullptr**

vector<Book> books;

vector<User> users;

vector<Admin> admins;

User \*current\_user = nullptr;

Admin \*current\_admin = nullptr;

**Operations**

* Add a book: input the information of a book, push into the **books** and record this operation to this admin.

A black screen with white text

AI-generated content may be incorrect.

Figure 1. Screenshot of adding a book

void adding() // 加一本書

{

string title, author;

int year, number;

cout << "Enter Book Name: ";

cin >> title;

cout << "Enter Published Year: ";

cin >> year;

cin.ignore();

cout << "Enter Book Author: ";

cin >> author;

cout << "Enter the Number of Available Copies: ";

cin >> number;

cin.ignore();

books.push\_back(Book(title, year, author, number));

current\_admin->record.push\_back(Record("Add ",title,year,author));

cout << "The book has been added to the system.\n";

}

* Search by the year: search the book by the year, push all the books published in the specific year by **searching\_year** into **found** and print all the books.

A black background with white text

AI-generated content may be incorrect.

Figure 2. Screenshot of searching books by year

void searching() // 用年份找書

{

cout << "Enter Year: ";

int year;

cin >> year;

cin.ignore();

vector<Book> found = searching\_year(year);

if(found.empty()) cout << "There is no book published in " << year << ".\n";

else

{

cout << "Name\tAuthor\tYear\tNumber of Available Copies\n";

for(auto &i : found)

cout << i.title << "\t" << i.author << "\t" << i.year << "\t" << i.number << "\n";

}

}

* **searching\_year**: find the books in the specific year and return them

vector<Book> searching\_year(int year) // 用年份去找要的書

{

vector<Book> result;

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

if(books[i].year == year)

result.push\_back(books[i]);

sorting\_title(result); // return之前先sort一下

return result;

}

* **sorting\_title**: apply **insertion sort** on books to satisfy lexicographical order. It works by iterating from the second element (index 1) to the end, taking each element as a “key,” and inserting it into its correct position within the already-sorted prefix by shifting larger elements one position to the right.

void sorting\_title(vector<Book> &result) // 用insertion sort排字典序

{

for (int i = 1; i < result.size(); ++i)

{

Book key = result[i];

int j = i - 1;

while (j >= 0 && result[j].title > key.title)

{

result[j + 1] = result[j];

j--;

}

result[j + 1] = key;

}

}

* Check out a book: enter the title, decrease the number of copies if it is greater than 0, and record this operation to this user/admin.

A black background with a black square

AI-generated content may be incorrect.

Figure 3. Screenshot of checking out a book

void checking() // 借書

{

string title;

cout << "Book Name: ";

cin >> title;

int idx = finding\_book(title);

if(idx == -1)

{

cout << "The Book can't be found.\n";

return;

}

else if(books[idx].number <= 0)

{

cout << "All books have been lent out\n";

return;

}

else

{

books[idx].number--;

cout << "Borrow Successfully!\n";

if(current\_user) current\_user->record.push\_back(Record("Check out", books[idx].title, books[idx].year, books[idx].author));

if(current\_admin) current\_admin->record.push\_back(Record("Check out", books[idx].title, books[idx].year, books[idx].author));

}

}

* Return a book: enter the title, increase number of copies if the book exists, and record this operation to this admin/user.

A black background with a black square

AI-generated content may be incorrect.

Figure 4. Screenshot of returning a book

void returning() // 還書

{

string title;

cout << "Book Name: ";

cin >> title;

int idx = finding\_book(title);

if(idx == -1)

{

cout << "Can't find this book.\n";

return;

}

else

{

books[idx].number++;

cout << "Return Successfully!\n";

if(current\_user) current\_user->record.push\_back(Record("Return ", books[idx].title, books[idx].year, books[idx].author));

if(current\_admin) current\_admin->record.push\_back(Record("Return ", books[idx].title, books[idx].year, books[idx].author));

}

}

* List all the books: use **sorting\_title** (already introduced above )to sort books and print them out if it’s not empty.

A black background with white text

AI-generated content may be incorrect.

Figure 5. Screenshot of listing all the books

void listing() // 列出所有書 一樣按照字典序

{

if(books.empty()) cout << "There is no any book.\n";

else

{

vector<Book> copy = books;

sorting\_title(copy);

cout << "Name\tAuthor\tYear\tNumber of Available Copies\n";

for(auto &i : copy)

cout << i.title << "\t" << i.author << "\t" << i.year << "\t" << i.number << "\n";

}

}

* Exit: if enter “5”, break the while loop and stop the system.

A black screen with white text

AI-generated content may be incorrect.

Figure 6. Screenshot of exiting the system

cout << "5. Exit.\n";

else if(num == 5) break;

**Main Functions**

進到系統後要先註冊帳號並登入系統，並將用戶分為user和admin兩種。(OE)

登入後可以進行以下功能：

1. Add: 只有admin有權限
2. Search: user / admin 皆有權限
3. Check out: user / admin 皆有權限
4. Return: user / admin 皆有權限
5. List: user / admin 皆有權限
6. Record: user / admin 皆有權限
7. Personnel management: 只有 admin 有權限
8. Log out: user / admin 皆有權限

**[P.S.]** Optimal Enhancement 功能的介紹在最後面。

A black screen with white text

AI-generated content may be incorrect.

Figure 7. Login interface

**A screen shot of a computer

AI-generated content may be incorrect.**

Figure 8. Admin Interface

A black screen with white text

AI-generated content may be incorrect.

Figure 9. User Interface

**Optimal Enhancement**

* **registering\_user**: register with username and identical passwords (user need to enter twice) and push it into **users**

void registering\_user() // user註冊

{

string username, password1, password2;

cout << "Enter Username: ";

cin >> username;

if(finding\_user(username) != -1 && finding\_admin(username) != -1)

cout << "The name has already exist. Try another.\n";

else

{

cout << "Enter password: ";

cin >> password1;

while(true)

{

cout << "Enter the password again: ";

cin >> password2; // 密碼要確認一次

if(password1 == password2)

{

users.push\_back(User(username, password1)); // 建立新用戶

cout << "User " << username << " has been created successfully.\n";

return;

}

else

cout << "Two passwords doesn't match, please try again.\n";

}

}

}

* **registering\_admin**: register with **adminname** and **identical passwords** (user need to enter twice) and push it into **admins**

void registering\_admin() // admin註冊

{

string adminname, password1, password2;

cout << "Enter Adminname: ";

cin >> adminname;

if(finding\_user(adminname) != -1 && finding\_admin(adminname) != -1)

cout << "The name has already exist. Try another.\n";

else

{

cout << "Enter password: ";

cin >> password1;

while(true)

{

cout << "Enter the password again: ";

cin >> password2; // 密碼要確認一次

if(password1 == password2)

{

admins.push\_back(Admin(adminname, password1)); // 建立新admin

cout << "Admin " << adminname << " has been created successfully.\n";

return;

}

else

cout << "Two passwords doesn't match, please try again.\n";

}

}

}

* **login\_user:** verify the user and change **current->user** into this user

bool login\_user() // user login

{

string username, password;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

int idx = finding\_user(username); // 找看看user是否存在

if(idx != -1 && users[idx].password == password)

{

current\_user = &users[idx];

cout << "Login successfully. Welcome, " << current\_user->username << "!\n";

return true;

}

else

{

cout << "Wrong username or password\n";

return false;

}

}

* **finding\_user**: find the position of an user

int finding\_user(const string &username) // 找user

{

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

if(users[i].username == username)

return i;

return -1;

}

* **login\_admin:** verify the admin and change **current->admin** into this admin

bool login\_admin() // admin login

{

string adminname, password;

cout << "Enter adminname: ";

cin >> adminname;

cout << "Enter password: ";

cin >> password;

int idx = finding\_admin(adminname); // 找看看admin是否存在

if(idx != -1 && admins[idx].password == password)

{

current\_admin = &admins[idx];

cout << "Login successfully. Welcome, " << current\_admin->adminname << "!\n";

return true;

}

else

{

cout << "Wrong username or password\n";

return false;

}

}

* **finding\_admin**: find the position of an admin

int finding\_admin(const string &adminname) // 找admin

{

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

if(admins[i].adminname == adminname)

return i;

return -1;

}

* **recording\_user**: output the record of an user

void recording\_user() // 記錄user opertaions

{

cout << "--- User Records for " << current\_user->username << " ---\n";

cout << "Operation" << "\t" << "Title" << "\t" << "Year" << "\t" << "Author\n";

for(auto &i : current\_user->record)

cout << i.operation << "\t" << i.title << "\t" << i.year << "\t" << i.author << "\n";

}

* **recording\_admin**: output the record of an admin

void recording\_admin() // 記錄admin的operations

{

cout << "--- Admin Records for " << current\_admin->adminname << " ---\n";

cout << "Operation" << "\t" << "Title" << "\t" << "Year" << "\t" << "Author\n";

for(auto &i : current\_admin->record)

cout << i.operation << "\t" << i.title << "\t" << i.year << "\t" << i.author << "\n";

}

* personnel management – upgrade users: list all the users at first, move the user from users to admins to evaluate permissions

void upgrade\_user()

{

cout << "Users: ";

for(auto &u : users)

cout << u.username << " ";

cout << "\n";

cout << "Promote user: ";

string name;

cin >> name;

int i = finding\_user(name);

if(i < 0)

{

cout << "No such user.\n";

return;

}

admins.emplace\_back(users[i].username, users[i].password);

users.erase(users.begin()+i);

cout << "Promoted.\n";

}

* personnel management – downgrade admins: list all the admins at first, move the admin from admins to users to reduce permissions

void downgrade\_admin()

{

cout << "Admins: ";

for(auto &i : admins)

cout << i.adminname << " ";

cout << "\n";

cout << "Demote admin: ";

string name;

cin >> name;

int i = finding\_admin(name);

if(i < 0)

{

cout << "No such admin.\n";

return;

}

users.emplace\_back(admins[i].adminname, admins[i].password);

admins.erase(admins.begin()+i);

cout << "Demoted.\n";

}

* personnel management – delete users: delete the user and erase it in users

void deleting\_user()

{

cout << "Users: ";

for(auto &u : users)

cout << u.username << " ";

cout << "\n";

cout << "Delete user: ";

string name;

cin >> name;

int i = finding\_user(name);

if(i < 0)

{

cout << "No such user.\n";

return;

}

users.erase(users.begin() + i);

cout << "Deleted.\n";

}

* log out: change the **current\_user->username** or **current\_admin->adminname** into **nullptr**

void logout\_user() // user logout

{

cout << "Logout: " << current\_user->username << "\n";

current\_user = nullptr;

}

void logout\_admin() // admin logout

{

cout << "Logout: " << current\_admin->adminname << "\n";

current\_admin = nullptr;

}