base.hpp

```cpp
#pragma once
#include <iconv.h>
#include <filesystem>
#include <algorithm>
#include <vector>
#include <fstream>
#include <iostream>
#include <conio.h>

using namespace std;

string utf8_to_gbk(const string &utf8_str)
{
    iconv_t cd = iconv_open("GBK", "UTF-8");
    if (cd == (iconv_t)-1)
        return "";
    size_t in_bytes_left = utf8_str.size();
    size_t out_bytes_left = in_bytes_left * 2;
    char *in_buf = const_cast<char *>(utf8_str.c_str());
    char out_buf[out_bytes_left];
    char *out_buf_start = out_buf;
    size_t ret = iconv(cd, &in_buf, &in_bytes_left, &out_buf_start,
&out_bytes_left);
    if (ret == (size_t)-1)
    {
        iconv_close(cd);
        return "";
    }
    *out_buf_start = '\0';
    iconv_close(cd);
    return string(out_buf);
}

string gbk_to_utf8(const string &gbk_str)
{
    iconv_t cd = iconv_open("UTF-8", "GBK");
    if (cd == (iconv_t)-1)
        return "";
    size_t in_bytes_left = gbk_str.size();
    size_t out_bytes_left = in_bytes_left * 2;
    char *in_buf = const_cast<char *>(gbk_str.c_str());
    char out_buf[out_bytes_left];
    char *out_buf_start = out_buf;
    size_t ret = iconv(cd, &in_buf, &in_bytes_left, &out_buf_start,
&out_bytes_left);
    if (ret == (size_t)-1)
    {
        iconv_close(cd);
        return "";
    }
    *out_buf_start = '\0';
    iconv_close(cd);
    return string(out_buf);
```

```
}

bool IsPureNumber(const string &input)
{
    return all_of(input.begin(), input.end(), ::isdigit);
}

class Base
{
public:
    virtual int Add() const = 0;
    virtual void Save() const = 0;
    virtual int Delete() const = 0;
    virtual int Edit() const = 0;
};
```

book.hpp

```
#define FILESYSTEM_BOOK "./data/book/"
#include "base.hpp"

class Book : public Base
{
public:
    string title;
    string author;
    string category;
    string keywords;
    string summary;
    int borrowTimes = 0;
    bool isBorrowed = false;

    Book(string Title = "", string Author = "", string Category = "", string Keywords = "", string Summary = "") : title(Title), author(Author), category(Category), keywords(Keywords), summary(Summary) {}
    Book(const Book &book) : title(book.title), author(book.author), category(book.category), keywords(book.keywords), summary(book.summary), borrowTimes(book.borrowTimes), isBorrowed(book.isBorrowed) {}
    ~Book() {}

    int Add() const override
    {
        string filePath = FILESYSTEM_BOOK + utf8_to_gbk(this->title) + ".txt";
        if (ifstream(filePath))
            return 0;
        else
        {
            ofstream file(filePath);
            if (!file)
                return -1;
            else
            {
                file << this->title << endl;
                file << this->author << endl;
```

```
                file << this->category << endl;
                file << this->keywords << endl;
                file << this->summary << endl;
                file << this->isBorrowed << endl;
                file << this->borrowTimes << endl;
                file.close();
                return 1;
            }
        }
    }

    void Save() const override
    {
        string filePath = FILESYSTEM_BOOK + utf8_to_gbk(this->title) + ".txt";
        ofstream file(filePath);
        file << this->title << endl;
        file << this->author << endl;
        file << this->category << endl;
        file << this->keywords << endl;
        file << this->summary << endl;
        file << this->isBorrowed << endl;
        file << this->borrowTimes << endl;
        file.close();
    }

    int Delete() const override
    {
        string filePath = FILESYSTEM_BOOK + utf8_to_gbk(this->title) + ".txt";
        if (remove(filePath.c_str()) == 0)
            return 1;
        else
            return -1;
    }

    int Edit() const override
    {
        ofstream file(FILESYSTEM_BOOK + utf8_to_gbk(this->title) + ".txt");
        if (!file)
            return -1;
        else
        {
            this->Save();
            file.close();
            return 1;
        }
    }

    friend ostream &operator<<(ostream &, const Book &);
};

ostream &operator<<(ostream &os, const Book &book)
{
    os << "书名：" << book.title << endl;
    os << "作者：" << book.author << endl;
```

```
    os << "分类：" << book.category << endl;
    os << "关键词：" << book.keywords << endl;
    os << "简介：" << book.summary << endl;
    if (book.isBorrowed)
        os << "借出状态：已借出" << endl;
    else
        os << "借出状态：未借出" << endl;
    os << "借出次数：" << book.borrowTimes << endl;
    return os;
}
```

user.hpp

```
#define FILESYSTEM_USER "./data/user/"
#include "base.hpp"

struct Record
{
    string bookName = "";
    string borrowTime = "";
    string returnTime = "";
    bool isReturned = false;
};

class User : public Base
{
public:
    string name;
    vector<Record> borrowRecords;
    int borrowTimes = 0;

    User(string Name = "") : name(Name) {}
    User(const User &user) : name(user.name), borrowRecords(user.borrowRecords), borrowTimes(user.borrowTimes) {}
    ~User() {}

    int Add() const override
    {
        string filePath = FILESYSTEM_USER + utf8_to_gbk(this->name) + ".txt";
        if (ifstream(filePath))
            return 0;
        else
        {
            ofstream file(filePath);
            if (!file)
                return -1;
            else
            {
                file.close();
                return 1;
            }
        }
    }

```

```
    void Save() const override
    {
        string filePath = FILESYSTEM_USER + utf8_to_gbk(this->name) +
".txt";
        ofstream file(filePath);
        for (auto record : this->borrowRecords)
        {
            file << record.bookName << endl;
            file << record.borrowTime << endl;
            file << record.returnTime << endl;
            file << record.isReturned << endl;
        }
        file.close();
    }

    int Delete() const override
    {
        string filePath = FILESYSTEM_USER + utf8_to_gbk(this->name) +
".txt";
        if (remove(filePath.c_str()) == 0)
            return 1;
        else
            return -1;
    }

    int Edit() const override
    {
        ofstream file(FILESYSTEM_USER + utf8_to_gbk(this->name) + ".txt");
        if (!file)
            return -1;
        else
        {
            this->Save();
            file.close();
            return 1;
        }
    }

    friend ostream &operator<<(ostream &, const User &);
};

ostream &operator<<(ostream &os, const User &user)
{
    os << "借阅次数：" << user.borrowTimes << endl;
    os << endl;
    os << "借阅记录：" << endl;
    os << endl;
    for (auto record : user.borrowRecords)
    {
        os << "书名：" << record.bookName << endl;
        os << "借书时间：" << record.borrowTime << endl;
        if (record.isReturned)
            os << "还书时间：" << record.returnTime << endl;
        else
            os << "还书时间：未还" << endl;
        os << endl;
```

```
    }
    return os;
}
```

bookmanager.hpp

```
#include "book.hpp"

class BookManager
{
public:
    Book getBook(const string &title) const
    {
        string filePath = FILESYSTEM_BOOK + utf8_to_gbk(title) + ".txt";
        if (!ifstream(filePath))
        {
            return Book();
        }
        else
        {
            ifstream file(filePath);
            if (!file)
            {
                return Book();
            }
            else
            {
                Book book;
                getline(file, book.title);
                getline(file, book.author);
                getline(file, book.category);
                getline(file, book.keywords);
                getline(file, book.summary);
                string line;
                getline(file, line);
                book.isBorrowed = (line == "1");
                getline(file, line);
                book.borrowTimes = stoi(line);
                file.close();
                return book;
            }
        }
    }

    vector<Book> searchBook(const string &keyword) const
    {
        vector<Book> results;
        for (const auto &entry :
filesystem::directory_iterator(FILESYSTEM_BOOK))
        {
            string filePath = entry.path().string();
            filePath = utf8_to_gbk(filePath);
            ifstream file(filePath);
            if (file)
            {
```

```
                Book book;
                getline(file, book.title);
                getline(file, book.author);
                getline(file, book.category);
                getline(file, book.keywords);
                getline(file, book.summary);
                string line;
                getline(file, line);
                book.isBorrowed = (line == "1");
                getline(file, line);
                book.borrowTimes = stoi(line);
                file.close();
                if (book.title.find(keyword) != string::npos ||
                    book.author.find(keyword) != string::npos ||
                    book.category.find(keyword) != string::npos ||
                    book.keywords.find(keyword) != string::npos ||
                    book.summary.find(keyword) != string::npos)
                {
                    results.push_back(book);
                }
            }
        }
        return results;
    }

    vector<Book> tenHotBooks() const
    {
        vector<Book> results;
        for (const auto &entry :
filesystem::directory_iterator(FILESYSTEM_BOOK))
        {
            string filePath = entry.path().string();
            filePath = utf8_to_gbk(filePath);
            ifstream file(filePath);
            if (file)
            {
                Book book;
                getline(file, book.title);
                getline(file, book.author);
                getline(file, book.category);
                getline(file, book.keywords);
                getline(file, book.summary);
                string line;
                getline(file, line);
                book.isBorrowed = (line == "1");
                getline(file, line);
                book.borrowTimes = stoi(line);
                file.close();
                if (book.borrowTimes > 0)
                {
                    results.push_back(book);
                }
            }
        }
        sort(results.begin(), results.end(), [](Book a, Book b)
             { return a.borrowTimes > b.borrowTimes; });
```

```
        if (results.size() > 10)
        {
            results.resize(10);
        }
        return results;
    }

    int deleteAllBooks() const
    {
        for (const auto &entry :
filesystem::directory_iterator(FILESYSTEM_BOOK))
        {
            string filePath = entry.path().string();
            filePath = utf8_to_gbk(filePath);
            remove(filePath.c_str());
        }
        return 1;
    }
};
```

usermanager.hpp

```
#include "user.hpp"

class UserManager
{
public:
    User getUser(const string &name) const
    {
        string filePath = FILESYSTEM_USER + name + ".txt";
        filePath = utf8_to_gbk(filePath);
        if (!ifstream(filePath))
        {
            return User();
        }
        else
        {
            ifstream file(filePath);
            if (!file)
            {
                return User();
            }
            else
            {
                User user(name);
                string line;
                while (getline(file, line))
                {
                    Record record;
                    record.bookName = line;
                    getline(file, line);
                    record.borrowTime = line;
                    getline(file, line);
                    record.returnTime = line;
                    getline(file, line);
```

```
                    record.isReturned = (line == "1");
                    user.borrowRecords.push_back(record);
                }
                user.borrowTimes = user.borrowRecords.size();
                file.close();
                return user;
            }
        }
    }

    vector<User> searchUser(const string &keyword) const
    {
        vector<User> results;
        for (const auto &entry :
filesystem::directory_iterator(FILESYSTEM_USER))
        {
            string filePath = entry.path().string();
            filePath = utf8_to_gbk(filePath);
            ifstream file(filePath);
            if (file)
            {
                User user;
                user.name = gbk_to_utf8(filePath.substr(12, filePath.size()
- 16));
                string line;
                while (getline(file, line))
                {
                    Record record;
                    record.bookName = line;
                    getline(file, line);
                    record.borrowTime = line;
                    getline(file, line);
                    record.returnTime = line;
                    getline(file, line);
                    record.isReturned = (line == "1");
                    user.borrowRecords.push_back(record);
                }
                user.borrowTimes = user.borrowRecords.size();
                file.close();
                if (user.name.find(keyword) != string::npos)
                {
                    results.push_back(user);
                }
            }
        }
        return results;
    }

    vector<User> tenActiveUsers() const
    {
        vector<User> results;
        for (const auto &entry :
filesystem::directory_iterator(FILESYSTEM_USER))
        {
            string filePath = entry.path().string();
            filePath = utf8_to_gbk(filePath);
```

```
            ifstream file(filePath);
            if (file)
            {
                User user;
                user.name = gbk_to_utf8(filePath.substr(12, filePath.size() - 16));
                string line;
                while (getline(file, line))
                {
                    Record record;
                    record.bookName = line;
                    getline(file, line);
                    record.borrowTime = line;
                    getline(file, line);
                    record.returnTime = line;
                    getline(file, line);
                    record.isReturned = (line == "1");
                    user.borrowRecords.push_back(record);
                }
                user.borrowTimes = user.borrowRecords.size();
                file.close();
                if (user.borrowTimes > 0)
                {
                    results.push_back(user);
                }
            }
        }
        sort(results.begin(), results.end(), [](User a, User b)
             { return a.borrowTimes > b.borrowTimes; });
        if (results.size() > 10)
        {
            results.resize(10);
        }
        return results;
    }

    int deleteAllUsers() const
    {
        for (const auto &entry : filesystem::directory_iterator(FILESYSTEM_USER))
        {
            string filePath = entry.path().string();
            filePath = utf8_to_gbk(filePath);
            remove(filePath.c_str());
        }
        return 1;
    }
};
```

manager.hpp

```
#include "bookmanager.hpp"
#include "usermanager.hpp"

class Manager : public BookManager, public UserManager
```

```
{
public:
    string getCurrentDateTime() const
    {
        time_t now = time(0);
        tm *ltm = localtime(&now);
        char buffer[80];
        strftime(buffer, sizeof(buffer), "%Y-%m-%d %H:%M:%S", ltm);
        return buffer;
    }

    int borrowBook(const string &userName, const string &bookName) const
    {
        Book book = getBook(bookName);
        if (book.title == "")
            return 0;
        User user = getUser(userName);
        if (user.name == "")
            return -2;
        if (book.isBorrowed)
            return -1;
        Record record;
        record.bookName = bookName;
        record.borrowTime = getCurrentDateTime();
        record.isReturned = false;
        user.borrowRecords.push_back(record);
        user.borrowTimes = user.borrowRecords.size();
        user.Save();
        book.isBorrowed = true;
        book.borrowTimes++;
        book.Save();
        return 1;
    }

    int returnBook(const string &userName, const string &bookName) const
    {
        Book book = getBook(bookName);
        if (book.title == "")
            return 0;
        User user = getUser(userName);
        if (user.name == "")
            return -2;
        bool found = false;
        for (auto &record : user.borrowRecords)
        {
            if (record.bookName == bookName && !record.isReturned)
            {
                record.returnTime = getCurrentDateTime();
                record.isReturned = true;
                user.Save();
                book.isBorrowed = false;
                book.Save();
                return 1;
            }
        }
        return -1;
```

```
    }
};
```

gui.hpp

```
#include "manager.hpp"

class GUI : public Manager
{
public:
    void ShowMenu() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "                图书管理系统" << endl;
        cout << "---------------------------------------" << endl;
        cout << "1. 添加书籍             9. 图书借阅" << endl;
        cout << "2. 删除书籍             10. 图书归还" << endl;
        cout << "3. 查找书籍             11. 借阅记录" << endl;
        cout << "4. 编辑书籍             12. 十大热门书籍" << endl;
        cout << "5. 添加用户             13. 十大活跃用户" << endl;
        cout << "6. 删除用户             14. 删除所有书籍" << endl;
        cout << "7. 查找用户             15. 删除所有用户" << endl;
        cout << "8. 编辑用户             16. 退出" << endl;
        cout << "---------------------------------------" << endl;
        cout << endl;
        cout << "请选择操作：";
    }

    string RemoveBlank(const string &str) const
    {
        auto start = str.find_first_not_of(" \t\n\r\f\v");
        if (start == string::npos)
            return "";
        auto end = str.find_last_not_of(" \t\n\r\f\v");
        return str.substr(start, end - start + 1);
    }

    void DisplayBook(const Book &book) const
    {
        cout << book;
    }

    void AddBook() const
    {
        system("cls");
        Book book;
        cout << endl;
        cout << endl;
        cout << "添加书籍" << endl;
        cout << endl;
        cout << endl;
        cout << "请输入书名：";
        getline(cin, book.title);
```

```
51          book.title = RemoveBlank(book.title);
52          if (book.title.empty())
53          {
54              cout << endl;
55              cout << "书名不能为空" << endl;
56              cout << endl;
57              cout << "按任意键返回" << endl;
58              getch();
59              return;
60          }
61          else if (getBook(book.title).title == book.title)
62          {
63              cout << "书籍已存在" << endl;
64              cout << endl;
65              cout << "按任意键返回" << endl;
66              getch();
67              return;
68          }
69          cout << "请输入作者：";
70          getline(cin, book.author);
71          book.author = RemoveBlank(book.author);
72          cout << "请输入分类：";
73          getline(cin, book.category);
74          book.category = RemoveBlank(book.category);
75          cout << "请输入关键词：";
76          getline(cin, book.keywords);
77          book.keywords = RemoveBlank(book.keywords);
78          cout << "请输入简介：";
79          getline(cin, book.summary);
80          book.summary = RemoveBlank(book.summary);
81          cout << endl;
82          int result = book.Add();
83          switch (result)
84          {
85          case 0:
86              cout << "书籍已存在" << endl;
87              break;
88          case -1:
89              cout << "保存失败" << endl;
90              break;
91          case 1:
92              cout << "保存成功" << endl;
93              break;
94          }
95          cout << endl;
96          cout << "按任意键返回" << endl;
97          getch();
98      }
99
100     void DeleteBook() const
101     {
102         system("cls");
103         cout << endl;
104         cout << endl;
105         cout << "删除书籍" << endl;
106         cout << endl;
```

```
        cout << endl;
        cout << "请输入书名：";
        string title;
        getline(cin, title);
        title = RemoveBlank(title);
        cout << endl;
        if (title.empty())
        {
            cout << "书名不能为空" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        if (getBook(title).title.empty())
        {
            cout << "书籍不存在" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        Book book(title);
        cout << "确认删除？(y/n)";
        string c;
        getline(cin, c);
        cout << endl;
        if (c != "y")
        {
            cout << "取消删除" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        int result = book.Delete();
        switch (result)
        {
        case -1:
            cout << "删除失败" << endl;
            break;
        case 1:
            cout << "删除成功" << endl;
            break;
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void SearchBook() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "查找书籍" << endl;
```

```
        cout << endl;
        cout << endl;
        cout << "请输入搜索词（回车显示所有书籍）: ";
        string title;
        getline(cin, title);
        title = RemoveBlank(title);
        cout << endl;
        cout << "查询结果" << endl;
        cout << endl;
        vector<Book> books = searchBook(title);
        int result = books.size() == 0 ? 0 : 1;
        switch (result)
        {
        case 0:
            cout << "书籍不存在" << endl;
            break;
        case 1:
            for (int i = 0; i < books.size(); i++)
            {
                cout << "书籍" << i + 1 << endl;
                DisplayBook(books[i]);
                cout << endl;
            }
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void EditBook() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "编辑书籍" << endl;
        cout << endl;
        cout << endl;
        cout << "请输入书名: ";
        string title;
        getline(cin, title);
        title = RemoveBlank(title);
        cout << endl;
        if (title.empty())
        {
            cout << "书名不能为空" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        Book oldBook = getBook(title);
        int result = oldBook.title.empty() ? 0 : 1;
        switch (result)
        {
        case 0:
            cout << "书籍不存在" << endl;
```

```
            break;
        case 1:
            DisplayBook(oldBook);
            Book book;
            cout << endl;
            cout << "请输入新书名：";
            getline(cin, book.title);
            book.title = RemoveBlank(book.title);
            if (book.title.empty())
            {
                cout << endl;
                cout << "书名不能为空" << endl;
                cout << endl;
                cout << "按任意键返回" << endl;
                getch();
                return;
            }
            cout << "请输入新作者：";
            getline(cin, book.author);
            book.author = RemoveBlank(book.author);
            cout << "请输入新分类：";
            getline(cin, book.category);
            book.category = RemoveBlank(book.category);
            cout << "请输入新关键词：";
            getline(cin, book.keywords);
            book.keywords = RemoveBlank(book.keywords);
            cout << "请输入新简介：";
            getline(cin, book.summary);
            book.summary = RemoveBlank(book.summary);
            cout << endl;
            book.isBorrowed = oldBook.isBorrowed;
            book.borrowTimes = oldBook.borrowTimes;
            oldBook.Delete();
            int result = book.Edit();
            switch (result)
            {
            case -1:
                cout << "保存失败" << endl;
                break;
            case 1:
                cout << "保存成功" << endl;
                break;
            }
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void DisplayUser(const User &user) const
    {
        cout << user;
    }

    void AddUser() const
    {
```

```
275         system("cls");
276         User user;
277         cout << endl;
278         cout << endl;
279         cout << "添加用户" << endl;
280         cout << endl;
281         cout << endl;
282         cout << "请输入用户名：";
283         getline(cin, user.name);
284         user.name = RemoveBlank(user.name);
285         cout << endl;
286         if (user.name.empty())
287         {
288             cout << "用户名不能为空" << endl;
289             cout << endl;
290             cout << "按任意键返回" << endl;
291             getch();
292             return;
293         }
294         int result = user.Add();
295         switch (result)
296         {
297         case 0:
298             cout << "用户已存在" << endl;
299             break;
300         case -1:
301             cout << "保存失败" << endl;
302             break;
303         case 1:
304             cout << "保存成功" << endl;
305             break;
306         }
307         cout << endl;
308         cout << "按任意键返回" << endl;
309         getch();
310     }
311
312     void DeleteUser() const
313     {
314         system("cls");
315         cout << endl;
316         cout << endl;
317         cout << "删除用户" << endl;
318         cout << endl;
319         cout << endl;
320         cout << "请输入用户名：";
321         string name;
322         getline(cin, name);
323         name = RemoveBlank(name);
324         cout << endl;
325         if (name.empty())
326         {
327             cout << "用户名不能为空" << endl;
328             cout << endl;
329             cout << "按任意键返回" << endl;
330             getch();
```

```
            return;
        }
        if (getUser(name).name.empty())
        {
            cout << "用户不存在" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        User user(name);
        cout << "确认删除？(y/n)";
        string c;
        getline(cin, c);
        cout << endl;
        if (c != "y")
        {
            cout << "取消删除" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        int result = user.Delete();
        switch (result)
        {
        case 1:
            cout << "删除成功" << endl;
            break;
        case -1:
            cout << "删除失败" << endl;
            break;
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void SearchUser() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "查找用户" << endl;
        cout << endl;
        cout << endl;
        cout << "请输入用户名（回车显示所有用户）：";
        string name;
        getline(cin, name);
        cout << endl;
        cout << "查询结果" << endl;
        cout << endl;
        vector<User> users = searchUser(name);
        int result = users.size() == 0 ? 0 : 1;
        switch (result)
        {
```

```
        case 0:
            cout << "用户不存在" << endl;
            break;
        case 1:
            for (int i = 0; i < users.size(); i++)
            {
                cout << "用户" << i + 1 << ": " << users[i].name << endl;
                cout << endl;
            }
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void EditUser() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "编辑用户" << endl;
        cout << endl;
        cout << endl;
        cout << "请输入用户名: ";
        string oldname;
        getline(cin, oldname);
        oldname = RemoveBlank(oldname);
        cout << endl;
        if (oldname.empty())
        {
            cout << "用户名不能为空" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        User oldUser = getUser(oldname);
        int result = oldUser.name.empty() ? 0 : 1;
        switch (result)
        {
        case 0:
            cout << "用户不存在" << endl;
            break;
        case 1:
            cout << "请输入新用户名: ";
            User user;
            getline(cin, user.name);
            user.name = RemoveBlank(user.name);
            cout << endl;
            if (user.name.empty())
            {
                cout << "用户名不能为空" << endl;
                cout << endl;
                cout << "按任意键返回" << endl;
                getch();
                return;
```

```
            }
            oldUser.Delete();
            int result = user.Edit();
            switch (result)
            {
            case -1:
                cout << "保存失败" << endl;
                break;
            case 1:
                cout << "保存成功" << endl;
                break;
            }
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void BorrowBook() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "图书借阅" << endl;
        cout << endl;
        cout << endl;
        cout << "请输入书名：";
        string title;
        getline(cin, title);
        title = RemoveBlank(title);
        cout << endl;
        if (title.empty())
        {
            cout << "书名不能为空" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        cout << "请输入用户名：";
        string name;
        getline(cin, name);
        name = RemoveBlank(name);
        cout << endl;
        if (name.empty())
        {
            cout << "用户名不能为空" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        int result = borrowBook(name, title);
        switch (result)
        {
        case 0:
```

```
            cout << "书籍不存在" << endl;
            break;
        case -1:
            cout << "书籍已借出" << endl;
            break;
        case -2:
            cout << "用户不存在" << endl;
            break;
        case 1:
            cout << "图书借阅成功" << endl;
            break;
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void ReturnBook() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "图书归还" << endl;
        cout << endl;
        cout << endl;
        cout << "请输入书名：";
        string title;
        getline(cin, title);
        title = RemoveBlank(title);
        cout << endl;
        if (title.empty())
        {
            cout << "书名不能为空" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        cout << "请输入用户名：";
        string name;
        getline(cin, name);
        name = RemoveBlank(name);
        cout << endl;
        if (name.empty())
        {
            cout << "用户名不能为空" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        int result = returnBook(name, title);
        switch (result)
        {
        case 0:
            cout << "书籍不存在" << endl;
```

```
            break;
        case -1:
            cout << "未借此书籍" << endl;
            break;
        case -2:
            cout << "用户不存在" << endl;
            break;
        case 1:
            cout << "图书归还成功" << endl;
            break;
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void BorrowRecord() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "借阅记录" << endl;
        cout << endl;
        cout << endl;
        cout << "请输入用户名：";
        string name;
        getline(cin, name);
        name = RemoveBlank(name);
        cout << endl;
        if (name.empty())
        {
            cout << "用户名不能为空" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        User user = getUser(name);
        int result = user.name.empty() ? 0 : 1;
        switch (result)
        {
        case 0:
            cout << "用户不存在" << endl;
            break;
        case 1:
            DisplayUser(user);
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void TenHotBooks() const
    {
        system("cls");
        cout << endl;
```

```
        cout << endl;
        cout << "十大热门书籍" << endl;
        cout << endl;
        cout << endl;
        vector<Book> books = tenHotBooks();
        if (books.size() == 0)
        {
            cout << "无记录" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        for (int i = 0; i < books.size(); i++)
        {
            cout << "书籍" << i + 1 << endl;
            DisplayBook(books[i]);
            cout << endl;
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void TenActiveUsers() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "十大活跃用户" << endl;
        cout << endl;
        cout << endl;
        vector<User> users = tenActiveUsers();
        if (users.size() == 0)
        {
            cout << "无记录" << endl;
            cout << endl;
            cout << "按任意键返回" << endl;
            getch();
            return;
        }
        for (int i = 0; i < users.size(); i++)
        {
            cout << "用户" << i + 1 << ": " << users[i].name << endl;
            cout << "借阅次数: " << users[i].borrowTimes << endl;
            cout << endl;
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void DeleteAllBooks() const
    {
        system("cls");
        cout << endl;
```

```
        cout << endl;
        cout << "确认删除所有书籍？(y/n)";
        string c;
        getline(cin, c);
        cout << endl;
        if (c == "y")
        {
            int result = deleteAllBooks();
            switch (result)
            {
            case 1:
                cout << "删除成功" << endl;
                break;
            default:
                cout << "删除失败" << endl;
                break;
            }
        }
        else
        {
            cout << "取消删除" << endl;
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
    }

    void DeleteAllUsers() const
    {
        system("cls");
        cout << endl;
        cout << endl;
        cout << "确认删除所有用户？(y/n)";
        string c;
        getline(cin, c);
        cout << endl;
        if (c == "y")
        {
            int result = deleteAllUsers();
            switch (result)
            {
            case 1:
                cout << "删除成功" << endl;
                break;
            default:
                cout << "删除失败" << endl;
                break;
            }
        }
        else
        {
            cout << "取消删除" << endl;
        }
        cout << endl;
        cout << "按任意键返回" << endl;
        getch();
```

```
    }

    void Exit() const
    {
        exit(0);
    }

    void Error() const
    {
        cout << endl;
        cout << "无效输入，请重新输入" << endl;
    }
};
```

library.hpp

```
#include "gui.hpp"

enum Choice
{
    AddBook = 1,
    DeleteBook,
    SearchBook,
    EditBook,
    AddUser,
    DeleteUser,
    SearchUser,
    EditUser,
    BorrowBook,
    ReturnBook,
    BorrowRecord,
    TenHotBooks,
    TenActiveUsers,
    DeleteAllBooks,
    DeleteAllUsers,
    Exit
};

class Library : public GUI
{
public:
    void CheckDirectory()
    {
        if (!filesystem::exists(FILESYSTEM_BOOK))
            filesystem::create_directories(FILESYSTEM_BOOK);
        if (!filesystem::exists(FILESYSTEM_USER))
            filesystem::create_directories(FILESYSTEM_USER);
    }
};
```

main.cpp

```
#include "library.hpp"

```

```cpp
int main()
{
    Library library;
    bool error = false;
    system("chcp 65001");

    while (true)
    {
        library.CheckDirectory();
        library.ShowMenu();
        if (error)
            library.Error();

        string input;
        int choice;
        getline(cin, input);
        if (!IsPureNumber(input) ||
            input.empty() ||
            (choice = stoi(input)) > Exit || choice < AddBook)
        {
            error = true;
            continue;
        }

        switch (choice)
        {
        case AddBook:
            library.AddBook();
            break;
        case DeleteBook:
            library.DeleteBook();
            break;
        case SearchBook:
            library.SearchBook();
            break;
        case EditBook:
            library.EditBook();
            break;
        case AddUser:
            library.AddUser();
            break;
        case DeleteUser:
            library.DeleteUser();
            break;
        case SearchUser:
            library.SearchUser();
            break;
        case EditUser:
            library.EditUser();
            break;
        case BorrowBook:
            library.BorrowBook();
            break;
        case ReturnBook:
            library.ReturnBook();
            break;
```

```
        case BorrowRecord:
            library.BorrowRecord();
            break;
        case TenHotBooks:
            library.TenHotBooks();
            break;
        case TenActiveUsers:
            library.TenActiveUsers();
            break;
        case DeleteAllBooks:
            library.DeleteAllBooks();
            break;
        case DeleteAllUsers:
            library.DeleteAllUsers();
            break;
        case Exit:
            library.Exit();
        }
        error = false;
    }
    return 0;
}
```