294728485%Peter Watts%Echopraxia, Firefall%0%5

972858284%Peter Watts%Blindsight, Firefall%1%6

285638693%Alexander Dumas%The Man in the Iron Mask%0%3

123456345%Frank Herbert%Dune%0%5

987654321%George Orwell%1984%0%5

123456789%J. K. Rowling%Harry Potter and the Philosopher's Stone%1%5

123454321%Suzanne Collins%The Hunger Games%0%5

#include <iostream>

#include <fstream>s

#include <string>

#include <unordered\_map>

#include <algorithm>

using namespace std;

/\* helper methods \*/

namespace helper {

string upper\_string(const string &str)

{

string upper;

transform(str.begin(), str.end(), back\_inserter(upper), ::toupper);

return upper;

}

string::size\_type find\_str\_ci(const string &str, const string &substr)

{

return upper\_string(str).find(upper\_string(substr));

}

}

class Book

{

string isbn;

string author;

string title;

bool borrowed;

double fineRate;

public:

Book()

: isbn(nullptr), author(nullptr), title(nullptr), borrowed(false), fineRate(5.0) {}

Book(const string isbn, const string author, const string title)

: isbn(isbn), author(author), title(title), borrowed(false), fineRate(5.0) {}

string getISBN() const { return isbn; }

string getAuthor() const { return author; }

string getTitle() const { return title; }

bool isBorrowed() const { return borrowed; }

double getFineRate() const { return fineRate; }

void setAuthor(const string str) { author = str; }

void setTitle(const string str) { title = str; }

void setBorrowed(bool status) { borrowed = status; }

void setFineRate(double rate) { fineRate = rate; }

};

class Library

{

unordered\_map<string, Book\*> books;

public:

~Library()

{

for (auto &p : books)

delete p.second;

books.clear();

}

void saveToText(const string path) const

{

ofstream outf(path);

if (!outf.is\_open()) {

cout << "Error opening file: " << path << endl;

exit(1);

}

for (const auto &pair : this->books) {

const Book \*book = pair.second;

outf << book->getISBN() << "%"

<< book->getAuthor() << "%"

<< book->getTitle() << "%"

<< (book->isBorrowed() ? "1" : "0") << "%"

<< book->getFineRate() << endl;

}

cout << "Written " << books.size() << " entries\n";

outf.close();

}

void loadFromText(const string path)

{

ifstream inpf(path);

if (!inpf.is\_open()) {

cerr << "Error opening file: " << path << endl;

exit(1);

}

string line;

int numBooks = 0;

this->~Library();

while (getline(inpf, line)) {

++numBooks;

size\_t delimiterPos1 = line.find("%");

if (delimiterPos1 == string::npos) {

cout << "Error reading line " << numBooks << " of file: " << path << endl;

exit(1);

}

size\_t delimiterPos2 = line.find("%", delimiterPos1 + 1);

if (delimiterPos2 == string::npos) {

cout << "Error reading line " << numBooks << " of file: " << path << endl;

exit(1);

}

size\_t delimiterPos3 = line.find("%", delimiterPos2 + 1);

if (delimiterPos3 == string::npos) {

cout << "Error reading line " << numBooks << " of file: " << path << endl;

exit(1);

}

size\_t delimiterPos4 = line.find("%", delimiterPos3 + 1);

if (delimiterPos4 == string::npos) {

cout << "Error reading line " << numBooks << " of file: " << path << endl;

exit(1);

}

try {

string isbn = line.substr(0, delimiterPos1);

string author = line.substr(delimiterPos1 + 1, delimiterPos2 - delimiterPos1 - 1);

string title = line.substr(delimiterPos2 + 1, delimiterPos3 - delimiterPos2 - 1);

bool borrowed = (line.substr(delimiterPos3 + 1, delimiterPos4 - delimiterPos3 - 1) == "1");

double fine = stod(line.substr(delimiterPos4 + 1));

this->addBook(isbn, author, title);

if (borrowed) this->borrowBook(isbn, false);

this->updateFineRate(isbn, fine);

} catch (invalid\_argument e) {

cout << "Error reading line " << numBooks << " of file: " << path << endl;

exit(1);

}

}

inpf.close();

cout << "Loaded " << numBooks << " entries\n";

}

void addBook(const string isbn, string author, string title)

{

if (books.count(isbn)) {

cout << "ISBN " << isbn << " already exists\n";

return;

}

delete books[isbn];

books[isbn] = new Book(isbn, author, title);

}

void viewAllBooks() const

{

printf("\n%11s | %16s | %20s | %s\n", "ISBN", "Author", "Title", "Status");

for (const auto& entry : books) {

string isbn = entry.first;

string author = entry.second->getAuthor();

string title = entry.second->getTitle();

bool available = !entry.second->isBorrowed();

if (isbn.length() > 10)

isbn = isbn.substr(0, 7) + "...";

if (author.length() > 15)

author = author.substr(0, 12) + "...";

if (title.length() > 19)

title = title.substr(0, 16) + "...";

printf("%11s | %16s | %20s | %s\n", isbn.c\_str(), author.c\_str(), title.c\_str(), available ? "available" : "borrowed");

}

}

void updateBook(string isbn, string author, string title)

{

if (!books.count(isbn)) {

cout << "ISBN " << isbn << " not found\n";

return;

}

books[isbn]->setAuthor(author);

books[isbn]->setTitle(title);

}

void delBook(string isbn)

{

if (!books.count(isbn)) {

cout << "ISBN " << isbn << " not found\n";

return;

}

delete books[isbn];

books.erase(isbn);

}

void updateFineRate(string isbn, double rate)

{

if (!books.count(isbn)) {

cout << "ISBN " << isbn << " not found\n";

return;

}

books[isbn]->setFineRate(rate);

}

void borrowBook(const string isbn, bool flag = true)

{

if (books.count(isbn) && !books[isbn]->isBorrowed()) {

books[isbn]->setBorrowed(true);

if (flag) cout << "Borrowed ISBN " << isbn << endl;

}

else if (flag) cout << "ISBN " << isbn << " is not in catalog\n";

}

void returnBook(const string isbn)

{

if (books.count(isbn) && books[isbn]->isBorrowed()) {

books[isbn]->setBorrowed(false);

cout << "Returned ISBN " << isbn << endl;

}

else cout << "ISBN " << isbn << " is already in catalog\n";

}

void checkAvailability(string isbn)

{

if (books.count(isbn)) {

if (books[isbn]->isBorrowed())

cout << "ISBN " << isbn << " is currently borrowed\n";

else cout << "ISBN " << isbn << " is available for borrowing\n";

}

else cout << "ISBN " << isbn << " not found\n";

}

void searchBooks(const string searchTerm) const

{

cout << "\nSearch results:\n";

int srch\_cnt = 0;

for (const auto &pair : books) {

if (pair.second->getISBN() == searchTerm ||

helper::find\_str\_ci(pair.second->getAuthor(), searchTerm) != string::npos ||

helper::find\_str\_ci(pair.second->getTitle(), searchTerm) != string::npos) {

++srch\_cnt;

cout << " ISBN: " << pair.second->getISBN() << "\n"

<< " Author: " << pair.second->getAuthor() << "\n"

<< " Title: " << pair.second->getTitle() << "\n"

<< " Fine: " << pair.second->getFineRate() << " per day late\n"

<< " Status: " << (!pair.second->isBorrowed() ? "Available" : "Borrowed") << "\n\n";

}

}

cout << "Found " << srch\_cnt << " results\n";

}

double calcFine(const string &isbn, int lateDays)

{

if (!books.count(isbn)) {

cout << "ISBN " << isbn << " not found\n";

return -1;

}

return lateDays \* books[isbn]->getFineRate();

}

};

int main()

{

Library library;

typedef enum {

END = 0, LOAD, SAVE, ADD\_BOOK, VIEW\_ALL, UPDATE\_BOOK, DEL\_BOOK,

SEARCH\_BOOK, RETURN\_BOOK, BORR\_BOOK, CHECK\_AVL,

UPDATE\_FINE, CALC\_FINE

} choice\_t;

while (true) {

cout <<

"\nChoices:\n"

" Enter " << LOAD << " to LOAD existing data\n"

" Enter " << SAVE << " to SAVE data to file\n"

" Enter " << ADD\_BOOK << " to ADD a new book entry\n"

" Enter " << VIEW\_ALL << " to VIEW all books\n"

" Enter " << UPDATE\_BOOK << " to UPDATE a book\n"

" Enter " << DEL\_BOOK << " to REMOVE book a entry\n"

" Enter " << SEARCH\_BOOK << " to SEARCH books\n"

" Enter " << RETURN\_BOOK << " to RETURN a book\n"

" Enter " << BORR\_BOOK << " to BORROW a book\n"

" Enter " << CHECK\_AVL << " to CHECK availability\n"

" Enter " << UPDATE\_FINE << " to UPDATE book fine rate\n"

" Enter " << CALC\_FINE << " to CALCULATE late fines\n"

" Enter " << END << " to EXIT\n"

"Enter your choice: ";

/\* taking input to an enum \*/

choice\_t choice = (choice\_t) ({ int ch; cin >> ch; ch; });

cin.ignore();

switch (choice) {

case LOAD: {

cout << "\nWARNING:\n"

<< " This will overwrite current data in memory\n"

<< " You are recommended to save your data first\n"

<< "Enter Y to load, cancel otherwise: ";

char ch; cin >> ch;

if (ch == 'y' || ch == 'Y') {

cout << "Enter file path: ";

string path;

cin.ignore();

getline(cin, path);

library.loadFromText(path);

}

break;

}

case SAVE: {

cout << "Enter file path: ";

string path;

getline(cin, path);

library.saveToText(path);

break;

}

case ADD\_BOOK: {

cout << "Enter ISBN: ";

string isbn; getline(cin, isbn);

cout << "Enter author: ";

string author; getline(cin, author);

cout << "Enter title: ";

string title; getline(cin, title);

library.addBook(isbn, author, title);

break;

}

case VIEW\_ALL:

library.viewAllBooks();

break;

case UPDATE\_BOOK: {

cout << "Enter ISBN: ";

string isbn; getline(cin, isbn);

cout << "Enter updated author: ";

string author; getline(cin, author);

cout << "Enter updated title: ";

string title; getline(cin, title);

library.updateBook(isbn, author, title);

break;

}

case DEL\_BOOK: {

cout << "Enter ISBN: ";

string isbn; getline(cin, isbn);

library.delBook(isbn);

break;

}

case SEARCH\_BOOK: {

cout << "Enter search query: ";

string searchTerm; getline(cin, searchTerm);

library.searchBooks(searchTerm);

break;

}

case RETURN\_BOOK: {

cout << "Enter ISBN: ";

string isbn; getline(cin, isbn);

library.returnBook(isbn);

break;

}

case BORR\_BOOK: {

cout << "Enter ISBN: ";

string isbn; getline(cin, isbn);

library.borrowBook(isbn);

break;

}

case CHECK\_AVL: {

cout << "Enter ISBN: ";

string isbn; getline(cin, isbn);

library.checkAvailability(isbn);

break;

}

case UPDATE\_FINE: {

cout << "Enter ISBN: ";

string isbn; getline(cin, isbn);

cout << "Enter daily late fine: ";

double rate; cin >> rate;

library.updateFineRate(isbn, rate);

break;

}

case CALC\_FINE: {

cout << "Enter ISBN: ";

string isbn; getline(cin, isbn);

cout << "Enter late days: ";

int lateDays; cin >> lateDays;

double fine = library.calcFine(isbn, lateDays);

if (fine < 0) break;

cout << "Fine for " << lateDays <<" D late return: "

<< fine

<< endl;

break;

}

case END: exit(0);

default:

cout << "\nInvalid input, try again\n";

}

}

return 0;

}