StackOverdue

Background Information

Ever wondered why checking out the books at your local library always took so long? A quick peek at the management system's codebase revealed years of mismanaged pointers, virtually no documentation, and the homogenous use of only the vector data structure. It was horrible! The library staff put up with it for a long time, but one day, the developers accidentally deleted the whole application. With no backups! They weren't even using git!

Fortunately for you, this was the last straw for the librarians. It was time for a new era of library management software. The possibilities are endless. You're hired as the new developer. Armed with a firm knowledge of puns, data structures, and algorithms, you set forth to create **StackOverdue**. This time, you're definitely going to use git.

Basic Information

The system time is represented as a single positive integer that begins with value 1 on program startup. For every day that passes, this value goes up by 1. A book has a positive integer identifier, a title, an author, and a genre. No two books can have the same identifier. No two books can have the same title and the same author. A user account has a positive integer identifier and the name of its owner. No two user accounts can have the same identifier, but multiple users can have same name. This library holds only one copy of any particular book and contains many user accounts.

A user can have a maximum of 10 books checked out simultaneously. When a user checks out a book, the due date is 15 days after the current day. The user can also renew the book a maximum of 2 times after the initial checkout, with each renewal pushing the due date back 5 days. If a user does not return a book before or on the due date, the book is considered overdue. If a user has books that are overdue, they cannot check out any new book(s) or renew any currently checked out book(s), including the overdue one(s). The user must return the overdue book(s) to "unblock" their account. Note that every book has a popularity score. This score is the total number unique users that have checked out the book in the past. On program startup, assume that the many users who previously checked out a book are not known (except the single one provided in the accounts file explained later), but there is nevertheless a popularity score provided. As the program runs and new users check out books, the popularity scores update accordingly, increasing by 1 for every unique user checking out a given book. Therefore, if a user renews a checked out book or checks out a book that they've checked out before, this has no effect on the book's popularity score.

Based on the above, you can make two inferences. First, if a user checks outs a book, renews it any time before the due date, renews it again any time before the second due date, and returns it on the third due date, the user will have legally kept a book for 25 days, without it ever being considered overdue. Second, a user can return a book and check it out again immediately afterwards. There is nothing preventing this. This library has no notion of a user reserving a book or paying a fine for an overdue book, but you can imagine that this would be implemented in the future.

Books File Format

A books data file contains all of the library's books. The first line represents the number of books in the library (and therefore, the file). Every subsequent line represents a single book. For each book, each individual component of its data is separated by a pipe character ("|") and is ordered as shown below:

```
<Book ID>|<Title>|<Author>|<Genre>|<Popularity Score>
```

A sample books data file might look like:

```
5
1|The Great Catsby|Will. E. Fitzgarfield|Non-Fiction|34
2|Tequila Mockingbird|Tim Federle|Drama|12
5|A Clockwork Blue|Fleorge Cush Jr.|Mystery|100
7|Quantum Physics & Existentialism|Rick N. Morty|Science Fiction|67
8|The Chronicles of Barnia|Pope Michaels|Non-Fiction|0
```

Accounts File Format

An accounts data file contains all of the library's user accounts, and the books those users have currently checked out. The first line represents the number of user accounts in the library (and therefore, the file). After that first line, there are many blocks of lines, with each block representing a single user account. The first line in a block represents the base details of a single user, along with the number of books checked out by that user. The subsequent lines in that block represent the books checked out by that user. Note that the due dates provided are based on a valid system time after 1. For each user account, each individual component of its data is separated by a pipe character ("|") and is ordered as shown below:

```
<User Account ID>|<Name>|<Number of Checked Out Books>
<Book ID>|<Due Date>|<Number of Times Renewed>
<Book ID>|<Due Date>|<Number of Times Renewed>
...
```

A sample accounts data file (with the first line of each user account bolded for clarity) might look like:

```
4
1|Aimon Syzman|3
5|3|2
18|4|1
9|13|0
7|Nathaniel Hawthorne|1
20|1|2
8|Chat Noir|0
25|Barack Obama|2
12|5|1
85|9|0
```

Program Startup

The program accepts two optional arguments of the form:

```
$ ./StackOverdue <Books File Name> <Accounts File Name>
```

The program can be provided no arguments, one argument, or two arguments. The first provided argument (if any) is the books data file name, and the second provided argument (if any) is the accounts data file name. The order for these arguments is pre-determined. There are no default files to use if a particular argument is not provided. If a provided file does not exist, skip over it.

These files provide an initial "loading" step to populate the library's books and accounts. Assume that any provided books data files do not contain duplicate books with the same id or title/author, and any provided accounts data files do not contain duplicate accounts with the same id. Assume that all expected values are valid (e.g., a book cannot have been renewed 4 times, a due date cannot be below 1, no two users can have checked out the same book). However, if—while reading in an accounts data file—your program encounters a checked out book that does not exist in the library, skip that book and display the problematic book id to the user. Some sample program startups might look like:

```
$ ./StackOverdue
No books provided.
No accounts provided.
(...normal program flow)

$ ./StackOverdue books.data
Loading books from "books.data".
No accounts provided.
(...normal program flow)

$ ./StackOverdue books.data
Could not find file "books.data". Skipping.
No accounts provided.
(...normal program flow)

$ ./StackOverdue books.data accounts.data
Loading books from "books.data".
Could not find file "accounts.data". Skipping.
(...normal program flow)

$ ./StackOverdue books.data accounts.data
Loading library from "books.data".
Loading accounts from "accounts.data".
Could not find library book with ID# 1111111.
Could not find library book with ID# 232.
(...normal program flow)
```

Commands

Library administrators can interact with the StackOverdue system via a command-line like prompt that waits for a command, completes/fails the request, and provides a new waiting prompt. However, you may **not** assume that any input provided to your application is valid input. See the error scenarios and the relevant output below:

SCENARIO	**OUTPUT**
Invalid command	`Invalid command.`
Invalid input to command (criteria, time, etc.)	`Invalid value.`
Library has no books and book access is required	`No books in your library.`
Library has no accounts and account access is required	`No accounts in your library.`
Provided book id does not exist	`BookID# XXX not found.`
Provided account id does not exist	`AccountID# XXX not found.`

In all of these scenarios, abort the command and wait for a new one. Do **not** abort the program. The scenarios described above are not shown in the sample outputs below, but they must be properly addressed regardless. Note that where extra horizontal spacing exists in a command's output, you should use a tab character ('\t'). Your application accepts the following set of commands:

COMMAND	**DESCRIPTION**

BROWSE	Prints out short descriptions of all the books in the library, ordered by title, author, genre, book id, or popularity score. Title, author, and genre ordering is ascending alphabetical. Book id ordering is ascending numerical. Popularity score ordering is descending numerical.
	
	<u>Examples</u>
	`> BROWSE`
	`Enter the criteria to sort by. (title/author/genre/bookid/popularity)`
	`> title`
	`1. “A Clockwork Blue” by Fleorge Cush Jr. (BookID# 5) [Mystery]. CHECKED OUT (AccountID# 1).`
	`2. “Quantum Physics & Existentialism” by Rick N. Morty (BookID# 7) [Science Fiction]. AVAILABLE.`
	`3. “Tequila Mockingbird” by Tim Federle (BookID# 2) [Drama]. AVAILABLE.`
	`4. “The Chronicles of Barnia” by Pope Michaels (BookID# 8) [Non-Fiction]. AVAILABLE.`
	`5. “The Great Catsby” by Will E. Fitzgarfield (BookID# 1) [Non-Fiction]. CHECKED OUT (AccountID# 1).`

BOOK	Prints out a full description of a specified book. A checked out book also displays checkout details. If overdue, print an extra line indicating so.
	
	<u>Examples</u>
	`> BOOK`
	`Enter the book id.`
	`> 2222`
	`Title: “Tequila Mockingbird”`
	`Author: Tim Federle`
	`BookID#: 2222`
	`Genre: Fiction`
	`Popularity Score: 12`
	`AVAILABLE`
	
	`> BOOK`
	`Enter the book id.`
	`> 1`
	`Name: “The Great Catsby”`
	`Author: Will E. Fitzgarfield`
	`BookID#: 1`
	`Genre: Non-Fiction`
	`Popularity Score: 34`
	`Borrower AccountID#: 111`
	`Due Date: 9`
	`Times Renewed: 0`
	`OVERDUE`

SEARCH	Prints out short descriptions for all the books in the library containing a specified phrase in either the book’s title or author. The search should be case-sensitive and can include substrings of words. If no results exist, indicate so.
	
	<u>Examples</u>
	`> SEARCH`
	`Enter the criteria to search by. (title/author)`
	`> author`
	`Enter the search phrase.`
	`> Fed`
	`Search Results:`
	`1. “Tequila Mockingbird” by Tim Federle (BookID# 2) [DRAMA]. AVAILABLE.`
	
	`> SEARCH`
	`Enter the criteria to search by. (title/author)`
	`> author`
	`Enter the search phrase.`
	`> fed`
	`No search results found.`

ACCOUNTS	Prints out short descriptions of all the accounts in the library, including short descriptions of the books checked out by each one. The accounts should be ordered by name, account id, or the number of books checked out. Name ordering is ascending alphabetical. Account id ordering is ascending numerical. Number of checked out books ordering is descending numerical.
	
	<u>Examples</u>
	`> ACCOUNTS`
	`Enter the criteria to sort by. (name/accountid/checkouts)`
	`> name`
	`1. Aimon Syzman (AccountID# 111). 2 books checked out:`
	`    1. “Tequila Mockingbird” by Tim Federle (BookID# 2) [Drama].`
	`    2. “A Clockwork Blue” by Fleorge Cush Jr. (BookID# 5) [Mystery].`
	`2. Le Chat Noir (AccountID# 2222). No books checked out.`

ACCOUNT	Prints out a full description of a specified account, including full descriptions of the books checked out. Note that the full descriptions for the books do not include the borrower account id line, unlike the output for BOOK.
	
	<u>Examples</u>
	`> ACCOUNT`
	`Enter the account id.`
	`> 111`
	`Name: Aimon Syzman`
	`AccountID#: 111`
	`2 books checked out (1 overdue):`
	`    1.`
	`    Name: “The Great Catsby”`
	`    Author: Will E. Fitzgarfield`
	`    BookID#: 1`
	`    Genre: Fiction`
	`    Popularity Score: 34`
	`    Due Date: 9`
	`    Times Renewed: 0`
	`    OVERDUE`
	`    2.`
	`    Title: “Tequila Mockingbird”`
	`    Author: Tim Federle`
	`    BookID#: 2222`
	`    Genre: Fiction`
	`    Popularity Score: 12`
	`    Due Date: 15`
	`    Times Renewed: 1`

CHECKOUT	Checks out a specified book to a specified account. If the book is successfully checked out, provide a full description of the book. If the book is successfully checked out and the user has never checked out the book before, add 1 to the book's popularity score. If the user has 10 books already checked out, has any overdue books, or the book is already checked out by someone else, they cannot check out any books. Note that the full descriptions for the books do not include the borrower account id line, unlike the output for BOOK.
	
	<u>Examples</u>
	`> CHECKOUT`
	`Enter the account id.`
	`> 333`
	`Enter the book id.`
	`> 2222`
	`Book successfully checked out.`
	`Name: "The Great Catsby"`
	`Author: Will E. Fitzgarfield`
	`BookID#: 2222`
	`Genre: Fiction`
	`Popularity Score: 35`
	`Due Date: 16`
	`Times Renewed: 0`
	
	`> CHECKOUT`
	`Enter the account id.`
	`> 333`
	`Enter the book id.`
	`> 111`
	`Account already has 10 books checked out.`
	
	`> CHECKOUT`
	`Enter the account id.`
	`> 333`
	`Enter the book id.`
	`> 111`
	`Account has books overdue.`
	
	`> CHECKOUT`
	`Enter the account id.`
	`> 333`
	`Enter the book id.`
	`> 111`
	`Book already is checked out.`

RENEW	Renews all the books in a specified account, if possible. If the user has any overdue books, then none of the books can be renewed. If a book has already been renewed twice, it specifically cannot be renewed again.

	<u>Examples</u>
	`> RENEW`
	`Enter the account id.`
	`> 2222`
	`1 of 2 books successfully renewed.`
	`    1.`
	`    Name: “The Great Catsby”`
	`    Author: Will E. Fitzgarfield`
	`    BookID#: 1`
	`    Genre: Fiction`
	`    Popularity Score: 34`
	`    Due Date: 18`
	`    Times Renewed: 1`
	`    Book successfully renewed.`
	`    2.`
	`    Title: “Tequila Mockingbird”`
	`    Author: Tim Federle`
	`    BookID#: 2222`
	`    Genre: Fiction`
	`    Popularity Score: 12`
	`    Due Date: 15`
	`    Times Renewed: 2`
	`    Book already renewed twice.`
	
	`> RENEW`
	`Enter the account id.`
	`> 2222`
	`Account has books overdue.`

RETURN	Returns a specified book to the library, if it is already checked out. Indicate if the book was returned on time, or overdue by some period of time.
	
	<u>Examples</u>
	`> RETURN`
	`Enter the book id.`
	`> 111`
	`Book successfully returned by AccountID# 333 (on time).`
	
	`> RETURN`
	`Enter the book id.`
	`> 111`
	`Book successfully returned by AccountID# 333 (overdue by 4 days).`
	
	`> RETURN`
	`Enter the book id.`
	`> 111`
	`Book is not currently checked out.`

RECOMMEND	Generates a list of five book recommendations for a specified account. The algorithm for choosing these recommendations is based on the user's most read genre, second most read genre, and most read author of all the books that user has ever checked out. The most read genre and the second most read genre each yield two book recommendation slots, which are the books with the highest popularity scores from the same genres. The most read author yields one book recommendation slot, which is the book with the highest popularity score written by that same author. Recommendations should be chosen in the order of top genre, second top genre, and author. Recommendations cannot appear more than once in the list. If a book has previously been checked out by a user, it should not appear in the recommendations. If there is ever a tie in popularity score or the number of most read genres/authors, resolve this situation however you see fit. In order to hit the 2/2/1 requirement, you must keep filling out those slots such that all of these criteria are satisfied. In situations that do not yield enough book recommendations for specific slots (user has read all the books in a genre, author has only written one book, the only other book by the author was selected for a genre slot, etc.), you may go below the five recommendation requirement. Think about what other situations can result in this. Disclude empty sections from the recommendations, if necessary.
	
	<u>Examples</u>
	`> RECOMMEND`
	`Enter the account id.`
	`> 3435345`
	`You love Fiction. We recommend:`
	`1. “Tequila Mockingbird” by Tim Federle (BookID# 2222) [Fiction]. AVAILABLE.`
	`2. “Apples & Oranges” by Susan Boyle (BookID# 12) [Fiction]. CHECKED OUT (AccountID# 1).`
	`You love Esoteric French Poetry. We recommend:`
	`1. “Le Sandwich du Fromage” by Jacques Somelier (BookID# 666) [Esoteric French Poetry]. CHECKED OUT (AccountID# 666).`
	`2. “Pomme de Terre” by Jacques Somelier (BookID# 667) [Esoteric French Poetry]. CHECKED OUT (AccountID# 666).`
	`You love Barack Obama. We recommend:`
	`1. “Hope for All” by Barack Obama (BookID# 13) [Non-Fiction]. AVAILABLE.`
	
	`> RECOMMEND`
	`Enter the account id.`
	`> 3435345`
	`You love Fiction. We recommend:`
	`1. “Tequila Mockingbird” by Tim Federle (BookID# 2222) [Fiction]. AVAILABLE.`
	`You love Barack Obama. We recommend:`
	`1. “Hope for All” by Barack Obama (BookID# 13) [Non-Fiction]. AVAILABLE.`

	`> RECOMMEND`
	`Enter the account id.`
	`> 3435345`
	`No available recommendations.`

ADDB	Adds a book to the library with a specified title, author, and genre. If a book with the same (case-sensitive) title and author exists, then do not allow the book to be added. Note that for proper categorization, genres are case sensitive, as well.
	
	<u>Examples</u>
	`> ADDB`
	`Enter the new book’s title.`
	`> Happy Days`
	`Enter the new book’s author.`
	`> Applesauce Timmy`
	`Enter the new book’s genre.`
	`> Non-Fiction`
	`BookID# 2223 successfully created.`
	
	`> ADDB`
	`Enter the new book’s title.`
	`> Happy Days`
	`Enter the new book’s author.`
	`> Applesauce Timmy`
	`Book with this title and author already exists.`

REMOVEB	Removes a specified book from the library. If the book is currently checked out, the program must “force return” the book from the borrower’s account.
	
	<u>Examples</u>
	`> REMOVEB`
	`Enter the book id.`
	`> 1`
	`“The Great Catsby” by Will E. Fitzgarfield successfully removed.`
	
	`> REMOVEB`
	`Enter the book id.`
	`> 1`
	`Force returning book from AccountID# 111.`
	`“The Great Catsby” by Will E. Fitzgarfield successfully removed.`

ADDA	Creates a new account with the specified name for the user. Multiple users can share the same name.
	
	<u>Examples</u>
	`> ADDA`
	`Enter the new user’s name.`
	`> Aimon Syzman`
	`AccountID# 112 successfully created.`

REMOVEA	Removes an account with a specified account id. If the account has books checked out, the program must “force return” the books before removing the account.
	
	<u>Examples</u>
	`> REMOVEA`
	`Enter the account id.`
	`> 333`
	`“The Great Catsby” by Will E. Fitzgarfield (BookID# 1) force returned.`
	`“Tequila Mockingbird” by Tim Federle (BookID# 2) force returned.`
	`Aimon Syzman’s account successfully removed.`

SYSTEM	Prints diagnostic information about the system.
	
	<u>Examples</u>
	`> SYSTEM`
	`System time: 24.`
	`Number of books: 167.`
	`Number of overdue books: 14.`
	`Number of accounts: 58.`
	`Number of overdue accounts: 8.`

TIME	Enables time travel mode (clearly, this is a sophisticated program that uses time travel for good) and passes time in the system, accepting a positive number of days as input.
	
	<u>Examples</u>
	`> PASS_TIME`
	`Enter the number of days to time travel.`
	`> 8`
	`Travelled 8 days through time (45 --> 53).`

EXPORT	Exports the library’s books and user accounts to files. The books in the library will be exported to their own file and the accounts will be exported to their own file, based on the file format specified earlier. The user specifies the name of the books file and the name of the accounts file. If a file with a provided name already exists, then overwrite it. If the library contains no books or if there are no accounts, then simply generate empty files.
	
	<u>Example</u>
	`> EXPORT`
	`Enter the name for the books file. (This may overwrite a file)`
	`> books.data`
	`Enter the name for the accounts file. (This may overwrite a file)`
	`> accounts.data`
	`Books data successfully exported to “books.data”.`
	`Accounts data successfully exported to “accounts.data”.`

HELP	This command displays all of the available commands and their descriptions.
	
	<u>Example</u>
	`> HELP`
	`BROWSE: Provides the status of all the books.`
	`BOOK: Provides the status of a specific book.`
	`SEARCH: Searches for all books with a key-phrase in the title or author.`
	`ACCOUNTS: Provides account snapshots for all user accounts.`
	`ACCOUNT: Provides an account snapshot for a specific user account.`
	`CHECKOUT: Checks a book out to a user account.`
	`RENEW: Renews a checked out book. (Up to two times allowed)`
	`RETURN: Returns a checked out book.`
	`RECOMMEND: Generates a list of recommended books for a given user.`
	`ADDB: Adds a new book to the library.`
	`REMOVEB: Removes a specific book from the library.`
	`ADDA: Creates a new user account in the library.`
	`REMOVEA: Removes a specific user account from the library.`
	`SYSTEM: Provides diagnostic information about the system.`
	`TIME: Fast forwards the system time by a specified number of days.`
	`EXPORT: Exports the library’s books and accounts to files.`
	`HELP: Displays this help menu.`
	`EXIT: Exits the program.`

EXIT	This command exits the program.
	
	<u>Example</u>
	`> EXIT`
	`Thank you for using StackOverdue!`

Assignment Completion Phases

First, you will create a **program design**. You are responsible for looking through this specification and creating a design proposal outlining the classes you expect to have, the general functionality that these classes provide, the data members/structures that these classes require, and how your classes interact with one another. Be sure to note whether any classes have inheritance relationships. Your design explanation should feature little to no code. It should instead focus on the macro-elements of your proposed program. You may include UML diagrams. After I create your personal assignment repository, I will open up a GitHub Issue in the repository asking you to define the design of your project. You will give your design proposal there. After you submit your design proposal, I will give you feedback on your design choices. The sooner you submit your design, the sooner I will give you feedback. Feel free to take my suggestions or not, and revise your design accordingly. I will not be providing a design of my own. You will then proceed with your **program implementation**. You can start writing code before you get feedback, if you feel confident. Ultimately, this final assignment gives you the most freedom.

Provided Files

For this assignment, you are not given any starter code. You will be provided some sample books and accounts input files that can be used to test your program. You will also be given an executable file that is a working solution based on this specification. Because the executable's solution code was compiled on the Linux lab machines, the executable can only be run on the Linux machines with confidence. Running it on any other system may yield undefined behavior.

Things to Think About

From an architectural perspective, there is no one perfect way to do this assignment. Nevertheless, good design choices can make your solution modular, extensible, efficient, and understandable. Endeavor to strike a balance that makes sense. Think about the following:

- What classes do you need to implement all of the required functionality? Is there inheritance? Which classes interact with the others, and how? Understand has-a vs. is-a relationships.
- Which classes act merely as basic "data containers" and which classes are responsible for the logic of the program commands? Remember that a single class should handle a single logical set of responsibilities. Don't make any one class do too much heavy lifting. Be modular.
- Which data structures do you need to make use of? You have many data structure at your disposal now, so be sure choose them carefully. Justify your choices in the README.
- Look at the STL documentation! Learn how to sort using the algorithms library. Learn about how to use iterators to loop through a data structure's elements. Overload the istream and ostream operators! It makes stream reading and writing flexible. There's so a huge amount that the STL allows you to do. You just need to ask, explore, and research.
- Remember that your output format and style should match the given output format and style EXACTLY in order for you to receive full credit. Be sure to test your running program alongside the provided executable to make sure that yours conforms.
- Don't forget about memory management!

Submission Details

Your submission must include all of the header/source files (*.h/*.cpp) required for your program to properly compile and run. You must include a makefile that includes the sources to be compiled. The generated executable must be named **StackOverdue**. Feel free to adapt the makefile provided in Assignment #1. Do not submit any generated executables or object files (*.o). You must also update the README file with any guidance that someone looking at your project needs to and might like to know; this could include compilation instructions, overall descriptions of your classes and how they interact, problems overcome, etc. Make sure to adequately comment your class interface files and class implementation files. The header files should describe each function and indicate run-time complexity, if appropriate. The source files should explain any non-trivial algorithms in the implementation. Ultimately, these comments are for the ease of the code reader and user. Confirm that your code successfully compiles and runs on the Linux lab machines. See the **Assignment Guide Using Git & GitHub** for step-by-step information about how to submit your assignment via git and GitHub.

Due Dates

There are two due dates for this assignment. The due date for the program design proposal is **Saturday, November 25th by 11:59pm**. I will not look at any design proposals submitted after this date. The due date for the implementation portion of the assignment is **Wednesday, December 13th by 11:59pm**. These are hard deadlines, meaning that there will be little leniency with regard to lateness. For every day that you miss the deadline for the implementation portion, I will deduct 10% from the project's final grade. A sample solution will be provided some time after the deadline.

Grading

This assignment is worth 20 points of your final grade, and is broken down as follows:

Components	Percentage	Relevant Questions
Correctness	50%	Do each of your commands provide output as expected based on the input?
Design	25%	Does your program design proposal separate out logical units into classes? Does it employ data structures that make sense and/or solve your problems efficiently? Is memory managed effectively?
Documentation	15%	Does your README document provide users with adequate information about your program? Do you adequately comment your class interface files and class implementation files (when necessary)?
Style	10%	Is your coding style readable and consistent? Do you follow (to the best of your ability) the modified Google Style Guide?

If your program does not compile on the Linux machines, you will receive no points for the **Correctness** component. If you do not submit your design proposal before its deadline, you will receive one point off of the **Design** component.

Final Words

This assignment is the hardest one yet! Don't be discouraged. Discuss things over with your peers often. You may be tempted to plagiarize. Don't. I will catch you, just like I did the plagiarizers last semester. Start early. (You'll thank yourself later.) Good luck!