Library

Objectives

To better understand the use of inheritance and polymorphism in object oriented design and programming by

1. Designing classes which are derived from other classes
2. Implementing derived and base classes
3. Using these classes in an application

Background

Inheritance and polymorphism are two fundamental features of OO design and programming. They allow us to group related objects that allow us to share common behaviors yet provide specialized behavior when necessary. In this project you will be implementing base classes and derived classes and using virtual functions.

Project Description

We will once again simulate a simple library. The library will consist of a set of library books and set of library patrons (people who borrow and return books). Commands will be read from a command file.

Your program will read commands from a file to perform the following functions.

1. Change the date.   
   When the date changes, fines are calculated for each overdue book.
2. Add a book to the library
3. Add a patron to the library   
   When a patron is added to the library, he is assigned a unique library card number.
4. Check out a book to a patron
5. Allow a patron to return (check in) a book   
   When a libarary book is returned, the overdue book fee (if any) is paid.
6. List all books in the library
7. List all books in the library which are of a specified type
8. List all patrons of the library
9. List all adult patrons of the library
10. List all child patrons of the library
11. List all books currently checked out to a particular patron

|  |  |
| --- | --- |
|  | Notes |
|  |  |
|  |  |
|  | The description of the LibraryBook class states "It must use a pointer to the LibraryPatron who has checked out the book."  Using an actual pointer variable will not work in this instance. If a LibraryBook actually points to a LibraryPatron and the array of LibraryPatrons is reallocated, then the pointers in the checked out LibraryBooks are all "dangling". So when is a pointer not a pointer? When it's an index. Instead of using an actual pointer, try storing the patron's index into the LibraryPatron array. Since we're not deleting patrons and you're not shuffling the patrons when a new one is added, a patron's array index remains constant. Alternatively, the LibraryPatron's unique Library card number could also be kept.  As noted on the discussion board, there were some inconsistencies between the description of each class' print() method and the required outputs. A print() method is still required, but the specification of the print() methods have been removed. The output requirements have been updated to reflect all required data.  It is NOT necessary to dynamically allocate all data members and local variables in project 4. The only required dynamic allocation is for the arrays of LibraryBooks and LibraryPatrons in the Library class.  A note on overdue fine calculations so that we all do them the same....      If a book is checked out on day N for 7 days, it is due on day N + 6 and is overdue on day N + 7 |

Project Specification

Class Design Guidelines

Your program must include the C++ classes listed below. Some classes have special restrictions. The following restrictions apply to all classes.

1. All data in all classes must be private.
2. A default constructor, a copy constructor, an assignment operator and a destructor must be written for all classes, even ff the compiler's default behavior is sufficient.
3. The methods for the base classes are specified below. You MAY NOT add any additional public methods.
4. All allowable member functions may be overloaded.
5. As always, you are free to add whatever private methods and private data you deem necessary for any class.
6. All class designs must follow basic OO principles of encapsulation and data hiding. Reminder:It is poor design for library methods to be reading data from the input file.
7. No friends are permitted.

Required Classes

The Library Class

The Libarary class must contain the following data.

1. A dynamically allocated array of LibraryBooks. Initially, the array is allocated with enough space for one LibraryBook. The capacity of the array is doubled whenever the array becomes full.
2. A dynamically allocated array of LibraryPatrons. Initially, the array is allocated with enough space for one LibraryPatron. The capacity of the array is doubled whenever the array becomes full.

The library class provides public methods to support the operations listed above in the project description. In addition,

* The library assigns unique library card numbers to each new patron.   
  The first library card is number 1001.
* The library assess fines for overdue books at the beginning of each day   
  Adults are charged $0.25 per day per book   
  Children are charged $0.10 per day per book
* The library considers fines for a book as paid when the book is checked in
* Children (under 12) may check out books for only 7 days
* Adults (12 and over) may check out books for 14 days.

There is no need for an overloaded insertion operator for this class.

The Book Class

The book class is a **BASE CLASS** that contains the title, author, and type of the book. A book is uniquely identified by its title. The book class supports only constructors, accessors, mutators, a virtual destructor, and a virtual print( ) method. You may optionally write the operator<< function for the Book class.

LibraryBook class

The LibraryBook class publicly inherits from the Book class. Its design is up to you, but with the following requirements.

1. It must use the Date class to store the library book's due date.
2. It must use a "pointer" to the LibraryPatron who has checked out the book.
3. It must support a virtual print() function

The Person Class

The Person class is a **BASE CLASS** that contains the name and age of the person. The Person class supports only constructors, accessors, mutators, a virtual destructor, and a virtual print() method. An overloaded insertion operator may be provided.

The LibraryPatron class

The LibraryPatron class publicly inherits from the Person class. Its design is up to you, but at a minimum, it must support a print() method. A library card number uniquely identifies a LibraryPatron.

The Date class

Dates are an integral part of virtually every computer system. At a minumum dates (and times) are used to record when certain important events happen. Often date calculations are an important part of your applications.

* What day of the week is it?
* Is this a leap year? (By the way - do you know the "complete" definition of leap year?)
* How many days are there between two dates (subtraction)
* What day will it be in N days (addition)
* Many more

The obvious way to store a date (year, month, day) is not always the best (does anyone remember Y2K?) because of the difficulty it imposes on date cacluations. For example, suppose you wanted to know what day it is 500 days from now. Your calculation would have to consider issues such as crossing month and year boundaries and crossing leap years.

An often used alternative is to use an integer to store a "day number". Some date in the past (such as Jan 1, 1970) is chosen to be day #1. Every night at midnight, the integer is incremented. This approach makes adding and subtracting dates less problematic. This is the design you must implement. For our project, day 1 will be Tuesday, January 1, 2002, which was not a leap year. You may assume that our simulation will not cross into 2003. (Do you see a potential Y2K type problem with this design?)

The Date class must support the following operations. Other operations may be provided, but must be justified in your design.

* constructor(s) of your choice
* a method to change the day to a new specified day
* a print( ) method that outputs the date in WWW DD MMM YYYY format (eg. Tue 01 Jan 2002).

The insertion operator for the Date class is optional.

Other Requirements and Restrictions

* Your program must handle the following errors:
  1. Incorrect number of command line arguments
  2. Command file was not opened successfully.

If either of the above errors occurs, send a message to the user (via cerr) and exit the program.

* 1. An attempt to add a duplicate book (same title)
  2. An attempt to check out a book that's already checked out
  3. An attempt to check out a book that does not exist in the library
  4. An attempt to check in a book that's not checked out
  5. An attempt to list books for a non-existent patron
  6. An attempt to check out a book to a non-existent patron

If any of these errors occur, send a message to the user (via cerr) and continue your program.

* Note that since a library card number uniquely identifies a patron and is used for checking out books, duplicate patron names are permissible.
* All book titles and book types are case sensitive.
* Your program must print out each command and its data (if any) to cout as it is encountered in the file. This will serve as a trace of your program's execution.
* Your program is invoked as follows:
* Proj4 filename

where filename is the name of command file described below.

Assumptions

You may assume the following:

1. If the command file can be opened, it will be in the syntactically correct format described below.
2. Dates will never go backwards and will not cross into 2020.

For output formatting purposes only (not for storing data), you may assume

1. Book titles are no more than 20 characters
2. Patron and Author names are no more than 15 characters
3. Book types are no more than 10 characters

The Command File

The command file contains command words and data for each command where appropriate. To minimize the amount of work necessary for parsing the command file, each command word and data item will be on a separate line. Blank lines may occur anywhere within the file and should be ignored. The command file contains the following command words and data items.

Students are encouraged to test their programs with short command files that focus on one or two related commands. After all commands have been tested as independently as possible, create command files with many varied commands

All command words will be in UPPERCASE. The data for each command is listed below under the command word, in order.

1. ADDBOOK -- add a book to the library   
   book's title   
   book's author   
   book's type
2. ADDPATRON -- add a patron to the library   
   patron's name   
   patron's age
3. BOOKS -- print books in the library   
   all -- print all books in the library   
   OR   
   a book type - print all books of this type in the library.
4. CHECKOUT -- check a book out of the library   
   book's title   
   patron's library card number
5. CHECKIN -- return a book to the library   
   book's title
6. DAY - change the current day   
   new day number
7. PATRONSBOOKS -- list all books checked out to a patron   
   patron's library card number
8. PATRONS -- list all patrons of the library   
   all - list all patrons   
   OR   
   adults - list only adults patrons   
   OR   
   children - list only child patrons

Sample Output

There are three required outputs from this program

1. a list of books in the library
2. a list of all books currently checked out to a particular patron
3. a list of patrons of the library

No sample output is being provided. You may format your output in any neat, easy to read format you feel is appropriate as long as the required information listed below is present. Your program must handle all situations in which there is nothing to print (i.e. no books or no patrons).

1. List of books

This output must include the following information for each book.

1. Title
2. Author
3. Type
4. Status - checked in, checked out, or overdue
5. If checked out or overdue list the book's due date and the name of the patron who has borrowed the book
6. If overdue list the amount of the fine due

2. List of Books for a Patron

This output must include the following information

1. The patron's name, age and library card number
2. The title and author of each book checked out by this patron including those which are overdue and any fines owed by the patron for the book.

3. List of Patrons

This output is a neatly organized list of patron names.

Project Make File

You are required to submit a make file with this project. The grader should simply need to type "make Proj4" and your project should compile and link to an executable called Proj4. The grader should also have the ability to do a "make clean" and "make cleanest."