Assignment IV, Deadline: October 22nd

Assignment learning objectives: This assignment is a tool to measure your knowledge in the following topics: classes, functions, *dynamic arrays*, constructors (default, overloaded, and copy constructors), *destructors* and *overloaded operators.*

The assignment problem:

As we mentioned in the classroom, a class should use dynamic data if the amount of data we expect it to hold may vary widely from one to the other. For example, a customer may have one invoice in in their history, or hundreds. A college student may have checked out a dozen of books from the school's library or none. The simplest solution is to have a class that stores as much as it needs to, and no more, or at least not much more. A lending library is an example of this sort of situation. A user may have two dozen books checked out, but on any given day, most users will not have any. Therefore, there is no point allocating memory for the books these users have checked out. Instead, we will dynamically allocated memory for these books when used and deallocate the memory when the books are returned.

You will write a program including a User class that has a dynamic array to hold information about the books that User has checked out. We will assume that each book has a sticker with a bar code representing a unique number; we can store books by bar code number (similar to the library system). You application will also need to create a dynamic array to hold all the users who has checked out book. The following diagram is an overview of the applications, input and output files.

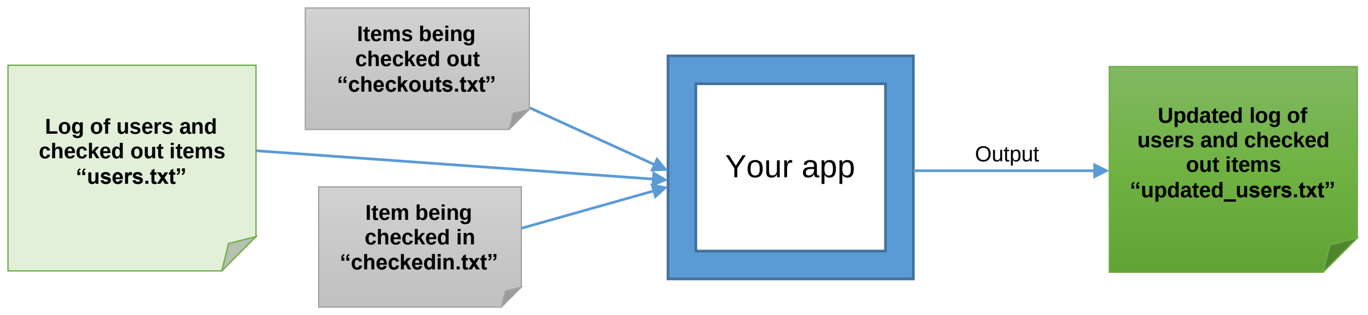
Items being checked out "checkouts.txt"
Log of users and checked out items "users.txt"
Item being checked in "checkedin.txt"
Your app
Output
Updated log of users and checked out items "updated_users.txt"

Figure1. Overview of the assignment's input and output data files

Your **application will have a *User* class with the following private data members:**

string for first and last name. *unsigned int* for the ID number. *unsigned int* for the number of books checked out and the array size. a string pointer (*string**) to manage the dynamic array.

The *User* class will have the following public methods:

Functions	Return type	Definition
Default constructor		creates new user with no parameters. The ID number defaults to 0 and the first and last names to empty strings
Overloaded constructor		initializes the new User's ID number, first and last names
`User(const User& other);`		copy constructor
`const User& operator=(const User& rhs);`	`User&`	overload the assignment operator
`GetIDNumber() const`	int	returns ID number, does not change object's data
`GetFirstName() const` `GetLastName() const`	string	returns first and last name, respectively, without changing object's data
`GetFullName() const`	string	returns a string consisting of the last name, then a comma, one space, and then the first name. Does not change the object's data.
`SetIDNumber(int NewNumber)`	void	assigns the parameter to the User's IDNumber if the parameter is in the range 100 <= NewNumber < 1000000; otherwise no changes are made and the function returns without making any modifications.
`Setters for FirstName and LastName`	void	assigns the parameters to the User’s first name and last name
`CheckoutCount() const`	int	returns the number of books the user has currently checked out. Does not change the object's data
`CheckOut(const string& BookIDCode);`	bool	verifies that the BookIDCode is not already checked out, and adds it to the dynamic array of checked-out books (adjusting array size if necessary). If the BookIDCode is already on the checked-out list, or for some reason the item cannot be added, it returns false; if adding the item was successful, this function returns true. This function may call the ResizeArray() private method (see below)
`CheckIn(const string& BookIDCode)`	bool	verifies that the BookIDCode is already on the checked-out list, and removes it. If this was the last item the user had checked out, the dynamic array is deleted. If the check-in was successful, this function returns true, otherwise it returns false.
`HasCheckedOut(const string& BookIDCode) const`	bool	returns true if the BookIDCode is on the User's checked-out list, otherwise returns false. Does not change the object's data.
`Clear()`	void	removes all of the object's data; the ID is set to 0, first and last name to empty strings, all checked-out data deleted, etc.
`~User()`		destructor

In addition, the *User* class has a *private* method: `bool ResizeArray().` This method is called when the dynamic array is full and another item is checked out. It creates a new, larger dynamic array; copies the checked-out data into it; deletes the old array; and makes the object's data pointer refer to the new array rather than the old one, updating the array size data member. This returns true if all operations are successful.

You application will also overload operators to be used on your defined type (i.e., User), as follows:

Operator	Definition
Stream extraction >>	`friend istream& operator >> (istream& in, User& item);` Used to read the object's data from an *istream* (read data from the text files). It is a friend function and takes a reference to an *istream* and a reference to a *User* as its parameters. It begins by calling *Clear()* on the object, then by reading in the data in the same format as the insertion operator >> writes it to a stream. Whether this operator makes an array just large enough for the checked-out items, or calls CheckOut() repeatedly, it is an implementation detail; handle it however you see fit.
Stream insertion <<	`friend ostream& operator<<(ostream& out, const User& item);` Used to send the object's data out to an *ostream* (write data in text files). It is a friend function that takes a reference to an *ostream* and a const reference to a *User* as its parameters. It writes the data to the stream in the following format: • The ID number, space, first name, space, last name, newline. • An integer with the number of items currently checked out, newline. • First item, space, second item, space, third item, space, etc… If the number of items checked out was 0, there is no third line of output.

Addition +	This is used as an alias for CheckOut. That is,
	`User Lisa;`
	`string item;`
	`Lisa = Lisa + item; // Lisa checks out an item`
	This function takes a *User* on the left, a *string* on the right, and returns a *const User*. It will be slightly easier to implement as a member function.
	Note that this should return a new User;
	`UserTemp = Lisa + item; // Lisa should not be modified by this`

Addition in place +=	Add in place: Another alias for CheckOut.
	`Phelps += item; // another way to check something out`
	This does not create a new object, it just checks it out for Phelps. This method does not return a value.

Comparison for equality == **and inequality !=**	Two objects of *User* type are considered the same if they have the same ID number, otherwise they are unequal.

The actual program:

You have three **input** files:

- The first file, *users.txt*, is the roster of all users who visited the library, including what books they have checked out (if any). Your program should use a dynamic array of Users class to store this data. (Whether you expand the array dynamically while the program is running, or read the file to count users, then read it again to do the actual input, is up to you.)

- The second file, *checkouts.txt*, lists what several users have checked out. It is in the form of a series of (UserID, ItemID) ordered pairs, with items separated by whitespace. Your program will process this file to check out the indicated items correspondingly.
- The third file, *checkedin.txt*, has several items that are being checked in. Search the data array until you find the user who has that item checked out first, and check it in using the class method.

Your application output:

Your application should produce a new text file, *updated_Users.txt*, which is an updated file with all check-outs and check-ins processed, in the same format as the *users.txt* file. Your program should also send error messages to cerr if you detect the following errors:

Checking out:

- No user can be found with this ID number: IDNum
- User IDNum has already checked out item ItemNum

Checking in:

- No user can be found with this item checked out: ItemNum
- This item is checked out by more than one user: (list all id numbers and check it in for ALL of them)

The following invariant rules should hold across all methods:

- If there are no items checked out, the object should have no dynamic data array.
- The size of the array is greater or equal to the number of items checked out.
- If a dynamic array is full and another item checked out, the array must be expanded. The new array should be twice as large as the old one.
- If there is no dynamic array and an item is checked out, make the dynamic array of size 5.
- If the array pointer is not nullptr, then it is pointing at validly-allocated memory; likewise, if the memory has been deleted (i.e. no items are checked out), the array pointer should be nulled.

Zip your project and upload it to blackboard no later than midnight on October 22nd.

~BestOfLuck()