Introduction

For this assignment, you will create a program to keep track of your books. You will be able to add books to the list, setting whether you have started reading them or not, and whether you own them or not. You can also update books that have previously been entered in the database.

Phase 1: Creating Book objects

First, create your **main.cpp** source file, as well as **Book.hpp** and **Book.cpp**

```
#ifndef _BOOK_HPP
#define _BOOK_HPP

#endif
```

About: Preventing duplicates

The purpose of the `#ifndef ... #define ... #endif` statement is so that, if multiple source code files are #including the same .hpp file, the C++ compiler won't mistakenly think that you've declared the same classes multiple times.

Enumerations: ReadingStatus and PurchaseStatus

Within **Book.hpp**, create two enumerations. Copy the following enumerations to the top of your **Book.hpp** file, but under the `#define`.

Our Book object will contain these two enumerations to keep track of each Book's status.

```
enum ReadingStatus { NOT_STARTED = 0, READING = 1, FINISHED = 2 };
enum PurchaseStatus { OWNED = 0, WISHLIST = 1 };
```

About: Enumerations

These are essentially ways that we can assign labels to integers, so each book can have a label like 0, 1, or 2, for Reading Status, and these number codes map to Not Started, Reading, and Finished. Enumerations can also be used like variables, such as creating a "ReadingStatus" variable:

```
ReadingStatus rs = NOT_STARTED;
```

Though technically any enumeration is really just an integer. You can also cast the enumeration directly to an integer, which we will do during our file read/write procedures.

```
int userInput = 2;
ReadingStatus rs = (ReadingStatus)userInput;
```

But these are just examples for now!

Book class

Our **Book** class will contain the title, author, and purchase and reading status. The class declaration will be in **Book.hpp**, below the enumeration declarations. Create a new class called Book.

The **Book** class should contain the following **private member variables**:

Data type	**Variable name**
string	m_title
string	m_author
ReadingStatus	m_readingStatus
PurchaseStatus	m_purchaseStatus

And the class will have the following **public member functions**. You will need to add the function declarations to **Book.hpp**, and the function definitions to **Book.cpp**.

In the declaration, you don't include the function body – only the function header, ended with a semi- colon. You also don't include "Book::" prior to the function name in the declaration – that's only for the definitions.

Class function declaration	**Class function definition**

`class Book`
`{`
`    public:`
`    string GetTitle();`
`};`
	`string Book::GetTitle()`
	`{`
	`    return m_title;`
	`}`

Common error: Missing ; Make sure that you have a semi-colon at the end of your class declaration, attached to the closing curly-brace!

Common error: Missing delcaration Every member function will have two parts, the declaration inside the class in the .hpp file, and the definition in the .cpp file.

Function specifications

void Book::DisplayBook()

Display the values of the following data, as well as labels for each item:

- `m_title`
- `m_author`
- The purchase status string (call GetPurchaseStatusString())
- The reading status string (call GetReadingStatusString())

The output once run should look something like this:

Example output

```
    Title:        Effective C++
    Author:       Scott Meyers
    Status:       Wishlist ,    Not started
```

---

string Book::GetAuthor()

Returns string

Return the value of `m_author`.

---

int Book::GetPurchaseStatus()

Returns int

Return the value of `m_purchaseStatus`.

---

string Book::GetPurchaseStatusString()

Returns string

If `m_purchaseStatus` is OWNED, then return the string ”Owned”. Or, if it is WISHLIST, then return the string ”Wishlist”.

---

int Book::GetReadingStatus()

Returns int

Return the value of `m_readingStatus`.

---

string Book::GetReadingStatusString()

Returns string

If `m_readingStatus` is NOT_STARTED, then return the string ”Not started”. Or if it is READING, then return the string ”Reading”. Or if it is FINISHED, then return the string ”Finished”.

---

string Book::GetTitle()

Returns string

Return the value of m_title.

---

void Book::SetBookInfo(string title, string author)

Parameters

Data type	**Variable name**
string	title
string	author

Use this function to set the private member variables, `m_title` and `m_author`, to the values from the parameters, `title` and `authors`.

---

void Book::SetPurchaseStatus(PurchaseStatus ps)

Parameters

Data type	**Variable name**
PurchaseStatus	ps

Set the private member variable, m_purchaseStatus, to the value passed in as `ps`.

---

void Book::SetReadingStatus(ReadingStatus rs)

Set the private member variable, m_readingStatus, to the value passed in as `rs`.

Parameters

Data type	**Variable name**
ReadingStatus	rs

Once you're done with the Book class declaration, it should look like this in **Book.hpp** :

```
#ifndef _BOOK_HPP
#define _BOOK_HPP

#include <iostream>
#include <string>
using namespace std;

enum ReadingStatus {  NOT_STARTED = 0, READING = 1, FINISHED = 2
    };
enum PurchaseStatus { OWNED = 0, WISHLIST = 1 };

class Book
{
public:
    void SetBookInfo( string title, string author );
    void SetReadingStatus( ReadingStatus rs );
    void SetPurchaseStatus( PurchaseStatus ps );
    void DisplayBook();

    string GetTitle();
    string GetAuthor();
    int GetReadingStatus();
    string GetReadingStatusString();
    int GetPurchaseStatus();
    string GetPurchaseStatusString();

private:
    string          m_title;
    string          m_author;
    ReadingStatus   m_readingStatus;
    PurchaseStatus  m_purchaseStatus;
};

#endif
```

and your function stubs in **Book.cpp** should look like this:

```
#include "Book.hpp"

void Book::SetBookInfo( string title, string author )
{
}

void Book::SetReadingStatus( ReadingStatus rs )
{
}

void Book::SetPurchaseStatus( PurchaseStatus ps )
{
}

void Book::DisplayBook()
{
}

string Book::GetTitle()
{
}

string Book::GetAuthor()
{
}

int Book::GetReadingStatus()
{
}

string Book::GetReadingStatusString()
{
}

int Book::GetPurchaseStatus()
{
}

string Book::GetPurchaseStatusString()
{
}
```

Testing phase 1

Before continuing, make sure your Book class works as intended. Within **main.cpp**, build out several test functions to make sure the functionality is working correctly.

It's just short-hand

Whenever a function is written like FunctionBlahBlah(...), the "..." are not literal arguments you're going to pass to the function; it is only short-hand to specify that the function requires some arguments, but without writing everything out.

Test the following:

1. Create a Book object, then call **SetBookInfo(...)**. Afterwards, call **GetTitle()** and **GetAuthor()**, and make sure the title and author values are returned properly.
2. Create a Book object, then use **SetPurchaseStatus(...)** to set a purchase status. Afterward, call GetPurchaseStatus() to make sure the integer form comes back correctly, and also call **GetPurchaseStatusString()** to make sure the string value matches.
3. Create a Book object, then use **SetReadingStatus(...)** to set a purchase status. Afterward, call **GetReadingStatus()** to make sure the integer form comes back correctly, and also call **GetReadingStatusString()** to make sure the string value matches.
4. Create a Book object, then call SetBookInfo(...), **SetReadingStatus(...)**, and **SetPurchaseStatus(...)**. After setting up the book, call **DisplayBook()** and make sure all the data has been stored and is being displayed.

Write tests and call the tests from main() like this:

```
#include <iostream>
using namespace std;
#include "Book.hpp"

void Phase1_Test1()
{
}

// ... and other tests ...

int main()
{
    Phase1_Test1();

    return 0;
}
```

Phase 2: Library - Dynamic array

Create two new files: **Library.hpp** and **Library.cpp** Make sure to begin with your `#ifndef ... #define ... #endif` statements. You will also want to `#include` the Book.hpp file.

We will start off by creating the Library class and its memory allocation code, to get it out of the way.

```
#ifndef _LIBRARY_HPP
#define _LIBRARY_HPP

#include "Book.hpp"

// ...

#endif
```

We will start off by creating the **Library** class and its memory allocation code, to get it out of the way.

The **Library** class should contain the following **private member variables**:

Data type	**Variable name**
Book*	m_bookList
int	m_arraySize
int	m_bookCount

And, for the time being, we will only have the following **public member functions**. Make sure to write the declaration inside the class in **Library.hpp**, and the definition in **Library.cpp**.

Function specifications

Library::Library()

The constructor prepares the program to be in a valid state. To do this:

1. Initialize **m_bookList** to nullptr
2. Initialize **m_arraySize** to 0 Initialize **m_bookCount** to 0.
3. Then, call the **AllocateMemory(...)** function and pass in 10 as the array size.

Library:: ˜Library()

The destructor must make sure that any memory allocated by this class is properly freed up before the class object is destroyed.

Just call the **DeallocateMemory()** function here.

---

void Library::DeallocateMemory()

Deallocate any memory that this class has allocated. To do this:

1. Check if **m_bookList** is pointing to nullptr. If it is NOT nullptr, then:
   (a) Delete the dynamic array being pointed to by **m_bookList**
   (b) Reset **m_bookList** to nullptr
   (c) Reset **m_arraySize** and **m_bookCount** to 0.

Creating a dynamic array

Remember how to create a dynamic array?

If we were making an array of integers, it would look like this:

```
int* numberArray;
numberArray = new int[ 100 ];
```

But in this case, you're working with `Book* m_bookList` , and a size passed in as the parameter.

void Library::AllocateMemory (int size)

Parameters

Data type	**Variable name**
int	size

Allocate space for the Book array using the **m_bookList** pointer. If space has already been allocated, then deallocate it and reallocate with the new size. Do the following:

1. If **m_bookList** is NOT nullptr, then call **DeallocateMemory()** first.
2. Afterward, allocate a Book array with the size given as a parameter, using the **m_bookList** pointer.
3. Set the **m_arraySize** member variable to the size passed in as the parameter.

Remember that `m_bookList` is a member variable the Library class. You should NOT be declaring a new `Book*` pointer variable here.

void Library::ResizeArray()

ResizeArray is responsible for creating a new dynamic array, copying over the values, and updating the pointer. Follow these steps to ”resize” the dynamic array:

1. Within the function, create a **Book*** pointer for a bigger array. It can be named **biggerArray**.
2. Using the biggerArray pointer, allocate space for a new Book array. Make its size the current array size `(m_arraySize) + 10`.
3. Use a for loop to iterate over items 0 through **m_arraySize**, copying each element FROM **m_bookList** TO **biggerArray**.
4. Free the small array - Delete the memory pointed to by **m_bookList**.
5. Update the pointer - Change **m_bookList** to point at the same address as **biggerArray**.
6. Update the array size - Change **m_arraySize** to be `m_arraySize + 10`.

```cpp
void Library::ResizeArray()
{
    Book* biggerArray;
    // Create a bigger array with the biggerArray pointer

    // Copy the data over from m_bookList to biggerArray
    for ( int i = 0; i < m_arraySize; i++ )
    {
    }

    // Delete the small array (m_bookList)

    // Update the pointer, point m_bookList to biggerArray

    // Update the size, add 10 to m_arraySize;
}
```

---

int Library::GetArraySize()

Returns int

Return the value of m_arraySize.

---

int Library::GetBookCount()

Returns int

Return the value of m_bookCount.

---

bool Library::IsArrayFull()

Returns bool

Return whether the array is full. You can figure this out by checking whether m_arraySize and m_bookCount are the same value.

If they're equal, then the array is full. If m_bookCount is less than m_arraySize, then the array is not yet full.

Testing phase 2

Again, before we continue, it is good to make sure your current functionality is working. Write more tests, and call them from **main()**

Remember to...

```
#include "Library.hpp"
```

in main.cpp!

Test the following:

1. Create a Library object, then call **AllocateMemory(...)**, passing in some size. Afterward, call **GetArraySize()** to make sure the size returned is the correct size.
2. Create a Library object, then call **AllocateMemory(...)**, passing in some size. Then, call **ResizeArray()**. After that, call **GetArraySize()** to make sure the size returned is the correct size. The size after resize should be the original size plus 10.

Additionally, you might set a breakpoint within your **DeallocateMemory()** function to make sure it is being called, and that the memory is being freed. You will lose points if your program has a memory leak.

Phase 3: Library - Program states

Now we will implement the main program. The library will allow the user to add new books, update existing books, view all books, and view the books stats. For the time being, the program won't save or load any data, so you will have to add books in each time you want to test.

Within **main()**, you will want to create a **Library** object, then call **Run()** on that object. This will let you test out the menus while you're working.

```
int main()
{
    Library library;
    library.Run();

    return 0;
}
```

Function specifications

void Library::DisplayBooksWithIndex()

Use a for loop, going from index 0 to **m_bookCount**, and display: The index The book's title - use **GetTitle()** The book's author - use **GetAuthor()**

```
for ( int i = 0; i < m_bookCount; i++ )
{
    // Do stuff to the m_bookList array
}
```

Calling GetTitle() and GetAuthor()

Remember that these functions belong to the Book object, so you will call it like:

`cout << m_bookList[ index ].GetTitle()` (and so on)

---

void Library::Run()

This function should call MainMenu.

(Note: Will be updated in a future phase)

---

void Library::MainMenu()

This function should create a program loop. Within this loop, you will display the main menu to the user and get their input. If their selection is to quit, then you will exit the loop.

Display the main menu:

Example output

```
----------------------------
| LIBRARY MAIN MENU |
---------------------

1.  Add new book
2.  Update book
3.  View stats
4.  View all books
5.  Save and quit

-----------------------
What do you want to do?
```

For option 1, call **NewBook()**

For option 2, call **UpdateBook()**

For option 3, call **ViewStats()**

For option 4, call **ViewAllBooks()**

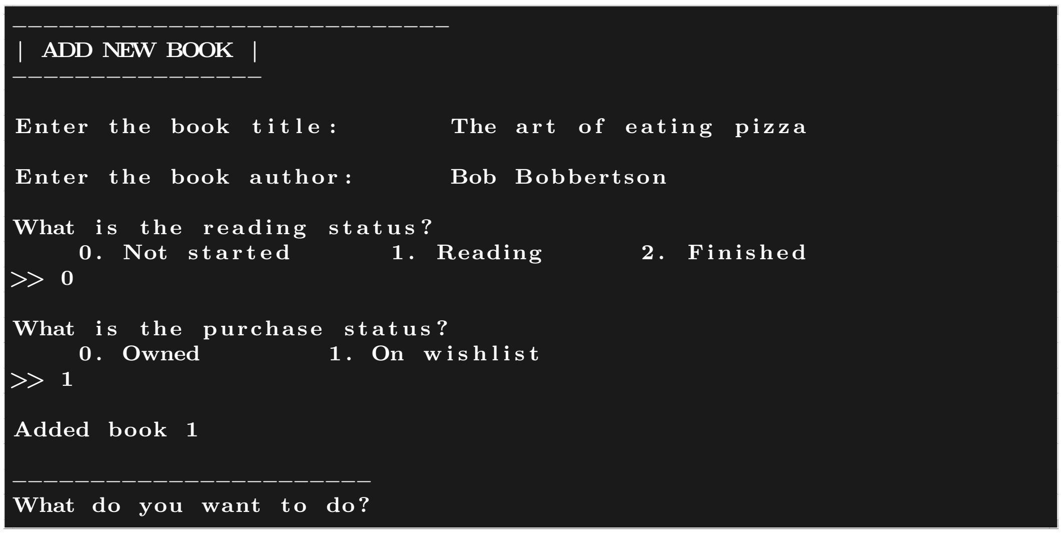
For option 5, tell the program loop to stop.

You can add additional options for Extra Credit features

Get the user's input, as an integer. Check what the input is, and call the appropriate function.

void Library::NewBook()

Example output



```
----------------------------
| ADD NEW BOOK |
----------------

Enter the book title:       The art of eating pizza

Enter the book author:      Bob Bobbertson

What is the reading status?
    0. Not started       1. Reading       2. Finished
>> 0

What is the purchase status?
    0. Owned        1. On wishlist
>> 1

Added book 1

-----------------------
What do you want to do?
```

This function will get data for the new book from the user, and then store it in a book element of the **m_bookList** array. Do the following:

1. Check to see if the array is already full. Use **IsArrayFull()** for this. If it IS full, then call **ResizeArray()** and then continue. Otherwise, if it is NOT full, just continue without doing anything.

2. Create four temporary variables to store user input: **title** and **author**, both strings, and **rs** and **ps**, both integers.

3. Ask the user for the book title, and store it in **title**.

4. Ask the user for the book author, and store it in **author**.

5. Ask the user what the reading status is.
   Display a menu with the options:
   `0. Not started 1. Reading 2. Finished`
   And store the user's choice in **rs**.

6. Ask the user what the purchase status is.
   Display a menu with the options:
   `0. Owned 1. On wishlist`
   And store the user's choice in **ps**.

7. For the next available element of **m_bookList**,
   call the following functions:
   (a) **SetBookInfo(...)**, passing in the **title** and **author**.
   (b) **SetReadingStatus(...)**, passing in **rs**, cast to a ReadingStatus
   (c) **SetPurchaseStatus(...)**, passing in **ps**, cast to a PurchaseStatus.
8. Increment **m_bookCount** by 1.

What is casting?

Casting is how we can change data from one data type to another data type. In this case, we're converting

int ↔ **PurchaseStatus** and **int** ↔ **ReadingStatus**.

When passing these as arguments, use:

`ReadingStatus( rs )` and `PurchaseStatus( ps )`

So, to set the current Book's reading status, use:

```
m_bookList[m_bookCount].SetReadingStatus(ReadingStatus(rs));
```

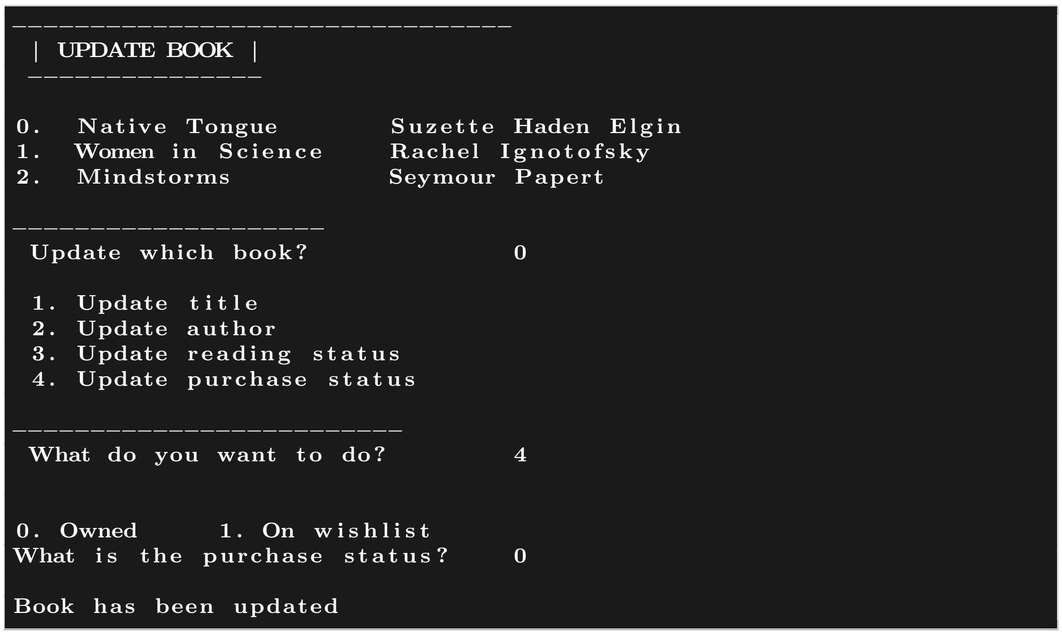
What is the next available element?

Our **m_bookCount** variable begins at 0, and each time we add a new book, it goes up by 1. Therefore, we can also use it as the next-available-index for **m_bookList**. So, the next available element is:

```
m_bookList[ m_bookCount ]
```

void Library::UpdateBook()

Example output



```
--------------------------------
 | UPDATE BOOK |
 ---------------

0.   Native Tongue        Suzette Haden Elgin
1.   Women in Science     Rachel Ignotofsky
2.   Mindstorms           Seymour Papert

--------------------
 Update which book?            0

 1. Update title
 2. Update author
 3. Update reading status
 4. Update purchase status

-------------------------
 What do you want to do?       4


0. Owned      1. On wishlist
What is the purchase status?   0

Book has been updated
```

The UpdateBook menu will allow the user to enter new information for books that have previous been added to the system.

1. Call the function **DisplayBooksWithIndex()** to display a list of all books.
2. Get the user's selection. The user needs to enter in an index from the array.
3. Display a menu for the user to ask what they want to update.
   (a) Title
   (b) Author
   (c) Reading status
   (d) Purchase status
4. Based on their choice, let the user enter a new title, author, reading status, or purchase status.
5. Afterward, update that book element with the new information. Use the appropriate function from the Book object

NOTE: For choices (3) and (4), make sure to display the list of purchase statuses, or reading statuses, for the user to choose from.

Book functions

Remember that the Book object contains the following functions:

- `SetInfo( title, author )`
- `SetReadingStatus( readingStatus )`
- `SetPurchaseStatus( purchaseStatus )`

To call these functions, you need to access the element from m_bookList, using the index that the user entered in step (2).

```
m_bookList[ index ].SetInfo( title, author );
```

Remember to cast rs and ps!

```
m_bookList[ index ].SetReadingStatus(ReadingStatus(rs));

m_bookList[ index ].SetPurchaseStatus(PurchaseStatus(ps));
```

void Library::ViewAllBooks()

Example output

```
------------------------------------
 | VIEW ALL BOOKS |
 ------------------

Native Tongue by Suzette Haden Elgin
Purchase Status: Owned
Reading Status:  Not started

Women in Science by Rachel Ignotofsky
Purchase Status: Wishlist
Reading Status:  Not started

Mindstorms by Seymour Papert
Purchase Status: Owned
Reading Status:  Finished
```

List out all the books in the array.

Use a for loop to iterate over indexes from 0 to **m_bookCount-1**.
Call **DisplayBook()** on each element of the array.

Calling DisplayBook()

DisplayBook() belongs to the Book object. You will need to access a specific book object thorugh the **m_bookList** array and the index given by the for loop...

```
m_bookList[ i ].DisplayBook();
```

---

void Library::ViewStats()

Example output

```
----------------------------
 | VIEW BOOK STATS |
 -------------------

Total books: 5

Books owned:        4
Books on wishlist:  1

Books finished:     2
Books reading:      1
Books not started:  2
```

The View Stats menu will display statistics on the books in the system, such as how many there are, how many have been read, how many are purchased, and more.

Create **integer** variables to store the total for the following items. Initialize each to 0.

- books owned
- books on wishlist
- books finished
- books currently reading
- books not yet started

Then, use a for loop to iterate over indexes 0 through **m_bookCount-1**. Within the loop:

- Use **GetPurchaseStatus()** to check the book's purchase status. Increment your "books owned" or "books on wishlist" variable by 1.
- Use **GetReadingStatus()** to check the book's reading stauts. Increment "books finished", "books currently reading", or "books not yet started" variable by 1.

Calling GetPurchaseStatus() and GetReadingStatus()

Once again, these two functions belong to the Book object.

You will need to access a specific book object thorugh the **m_bookList** array and the index given by the for loop...

```
m_bookList[ i ].GetPurchaseStatus()

m_bookList[ i ].GetReadingStatus()
```

Afterward, display the **m_bookCount**, as well as each of the 5 stats variables to the screen.

Testing phase 3

Before you continue, make sure all these features work properly. Write tests for the following:

1. Create a Library object, add 3 new books, then go to the View All Books screen and make sure they show up.
2. Create a Library object, add 15 new books, make sure that the array resizes at book #11 and the program doesn't crash.
3. Create a Library object, add 3 new books, then go to the Update Book menu. Make sure you're able to change each of the book's properties (title, author, reading status, and purchase status), and ensure all changes show up in the View All Books menu.
4. Create a Library object, add 5 new books, then run the Display Stats menu to make sure the stats are being displayed properly.

Phase 4: Library - Data saving and loading

Finally, we are going to save the book data to a text file, and re-load it in once the program opens. This way, you don't have to keep adding new books just to test everything.

Common error: Corrupted files

While working on these features, it is possible for you to write *bad data* out to the text file. In this case, your program might not run correctly.

You can double-check the text file to make sure it is in the correct format.

If something is wrong with it any you can't see anything wrong, try to erase the text file and start over.

You're going to be adding a SaveData() and LoadData() function to your program, as well as updating some existing functoins.

Function updates

void Library::Run()

In this function, update it so that **LoadData()** is called before **MainMenu()**

---

void Library::MainMenu()

In this function, update it so that **SaveData()** is called as the last line of the function, outside of the while loop.

Function specifications

void Library::SaveData()

We are going to make a standardized format to store the book data in.

Every book has four properties:

1. title
2. author
3. reading status
4. purchase status

We will store data in the file like this:

```
HEADER
TITLE
AUTHOR
READING
PURCHASE
```

Where each book will have 4 lines.

Follow these steps:

1. Create an **ofstream** object, with a name like **output** .
2. Open the filename, "books.txt", with the **output** object.
3. Create a for loop that iterates over the indexes from 0 to **m_bookCount - 1** . Within the loop:
   (a) Output "BOOK" and then the index number to the text file This will be all one word, like "BOOK0" and "BOOK1".

(b) Output the book's Title to the text file.

(c) Output the book's Author to the text file.

(d) Output the book's Reading Status (as an int) to the text file.

(e) Output the book's Purchase Status (as an int) to the text file.

Separate each field by a new line.

Book functions:

```
m_bookList[ i ].GetTitle()

m_bookList[ i ].GetAuthor()

m_bookList[ i ].GetPurchaseStatus()

m_bookList[ i ].GetReadingStatus()
```

Do a quick test by running the program, creating two books, then selecting the "Save and Quit" option from the menu.

Check the output file to make sure it lists 8 lines - both books.

```
BOOK0
Mindstorms
Seymour Papert
2
0
BOOK1
Native Tongue
Suzette Haden Elgin
2
0
```

fstream library

Make sure you have `#include <fstream>` at the top of your Library.cpp file!

void Library::LoadData()

Now we will need to read in the same text from the text file.

Create temporary variables to work with, while we're loading in data:

- `string header;`
- `string title;`
- `string author;`
- `int rs;`
- `int ps;`

Create an **ifstream** object named **input** , and open the "books.txt" file.

You will need to read the title and author with **getline(...)** functions, and you will use **input >>** for the header, reading status, and purchase status.

Common error: Input skipping

When you're switching between the >> and getline(...) methods of getting input, a line of input will be skipped between the last >> and the first getline(...).

To avoid this, make sure to put **input.ignore();** between the last >> and the first getline(...), where **input** is the name of the ifstream object.

Set up a while loop that will keep reading while the header is read successfully...

```
while ( input >> header )
{
}
```

Then, call **input.ignore();** , and read in the **title**, **author**, **rs**, and **ps**.

After these four lines are read, check to see if the array is full. Use **IsArrayFull()** . If the array IS full, then call **ResizeArray();** before continuing.

Then, using **m_bookList**[**m_bookCount**] , set the title, author, purchase status, and reading status with the functions **SetBookInfo(...)** , **SetPurchaseStatus(...)** , and **SetReadingStatus(...)**

Finally, add one to **m_bookCount**

Check the next page for more help.

```
void Library::SaveData()
{
    ofstream output( "books.txt" );

    for ( int i = 0; i < m_bookCount; i++ )
    {
        output << "BOOK" << i << endl;
        output << m_bookList[i].GetTitle() << endl;
        output << m_bookList[i].GetAuthor() << endl;
        output << m_bookList[i].GetReadingStatus() << endl;
        output << m_bookList[i].GetPurchaseStatus() << endl;
    }

    output.close();
}
```

```
void Library::LoadData()
{
    ifstream input( "books.txt" );

    string header;
    string title, author;
    int rs, ps;

    while ( input >> header )
    {
        input.ignore();
        getline( input, title );
        getline( input, author );
        input >> rs;
        input >> ps;

        if ( IsArrayFull() )
        {
            ResizeArray();
        }

        m_bookList[ m_bookCount ].SetBookInfo( title, author );
        m_bookList[ m_bookCount ].SetPurchaseStatus(
    PurchaseStatus( ps ) );
        m_bookList[ m_bookCount ].SetReadingStatus( ReadingStatus(
     rs ) );

        m_bookCount++;
    }

    input.close();
}
```

Testing phase 4

Make sure the save and load process works properly.

1. Remove the ”books.txt” file. Start the program, add several books, then save and quit. Make sure next time you run it, all books show up under the View All Books menu.

Extra credit

If you want to implement some extra credit features, you can do the following:

Export full list in human-readable format

Create an additional option to the Main Menu, and as a function in the Library class.

This function should create a **text** or **CSV** format file with all the books, but in a human-readable format.

This means that, instead of 0, 1, or 2 for reading status, you display "Not started", "Reading", or "Finished", and similarly for purchase status.

Export the wishlist

Export a list that contains only the books that are marked as "on wishlist" as their purchase status.

Export the reading list

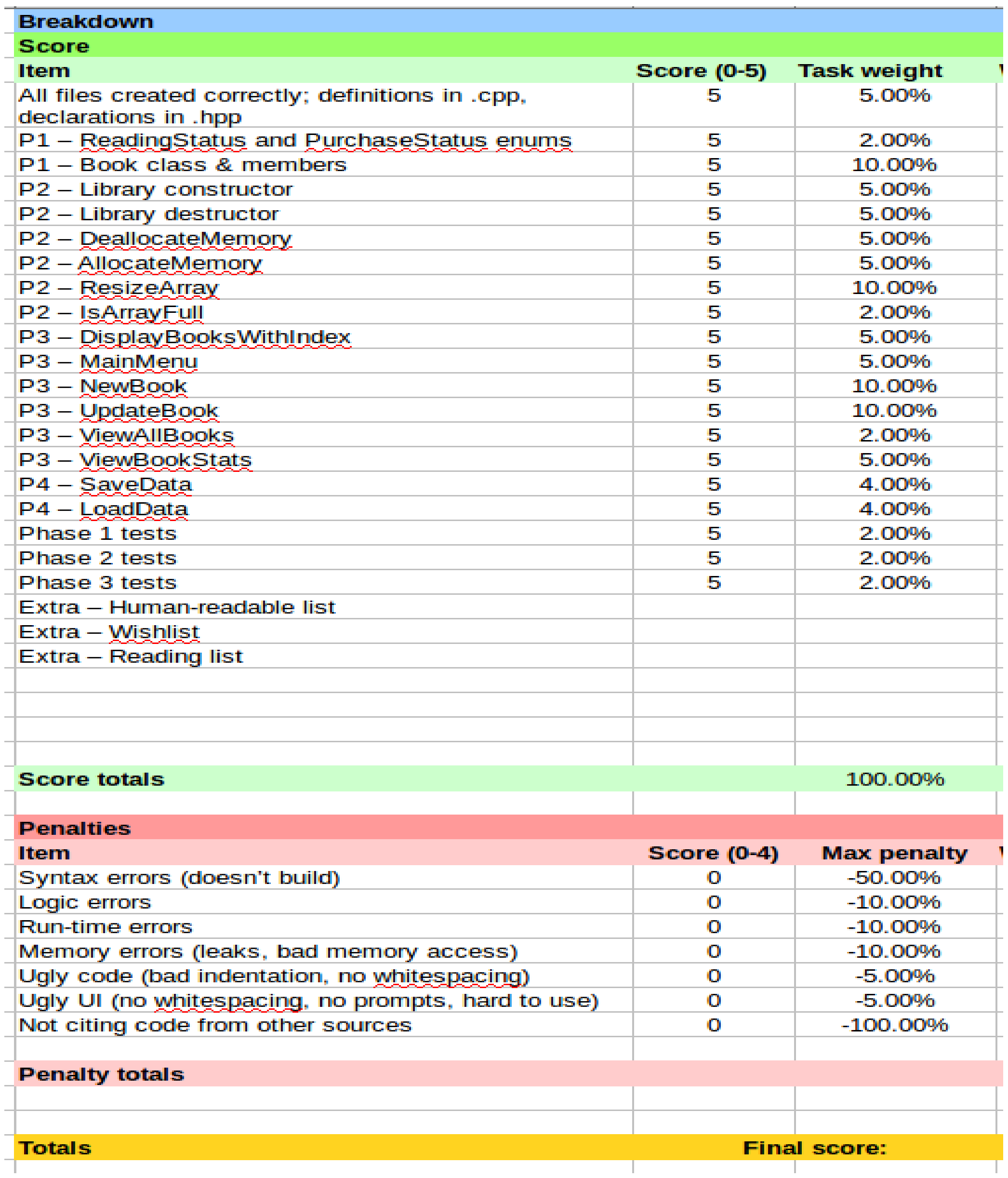
Export a list that contains only the books that are marked as "reading" or "not started" as their reading status; no complete books.

Turning in your project

All you need to turn in are the source files, **.cpp** and **.hpp** files. Make sure you turn in:

1. main.cpp
2. Book.hpp
3. Book.cpp
4. Library.hpp
5. Library.cpp

Grading breakdown



Breakdown		
Score		
Item	**Score (0-5)**	**Task weight**
All files created correctly; definitions in .cpp, declarations in .hpp	5	5.00%
P1 – ReadingStatus and PurchaseStatus enums	5	2.00%
P1 – Book class & members	5	10.00%
P2 – Library constructor	5	5.00%
P2 – Library destructor	5	5.00%
P2 – DeallocateMemory	5	5.00%
P2 – AllocateMemory	5	5.00%
P2 – ResizeArray	5	10.00%
P2 – IsArrayFull	5	2.00%
P3 – DisplayBooksWithIndex	5	5.00%
P3 – MainMenu	5	5.00%
P3 – NewBook	5	10.00%
P3 – UpdateBook	5	10.00%
P3 – ViewAllBooks	5	2.00%
P3 – ViewBookStats	5	5.00%
P4 – SaveData	5	4.00%
P4 – LoadData	5	4.00%
Phase 1 tests	5	2.00%
Phase 2 tests	5	2.00%
Phase 3 tests	5	2.00%
Extra – Human-readable list		
Extra – Wishlist		
Extra – Reading list		
Score totals		100.00%
Penalties		
Item	**Score (0-4)**	**Max penalty**
Syntax errors (doesn't build)	0	-50.00%
Logic errors	0	-10.00%
Run-time errors	0	-10.00%
Memory errors (leaks, bad memory access)	0	-10.00%
Ugly code (bad indentation, no whitespacing)	0	-5.00%
Ugly UI (no whitespacing, no prompts, hard to use)	0	-5.00%
Not citing code from other sources	0	-100.00%
Penalty totals		
Totals		**Final score:**