Fall Semester 2019

Library Publication Storage and Retrieval Application

V1.0(Final Validation Submission)

When Books and other publications arrive in a library, they should be tagged and put on shelves, so they are easily retrievable to be lent out to those who need it.

Your task is to design an application that receives the publications and stores them into the system with the information needed for their retrieval.
Later, each publication can be lent out to members of the library with a due date for return.

Before we start developing the application, we need to have few classes developed to help us with the dates in the system and also the user interface of the application.

CITATION AND SOURCES

When submitting the milestone deliverables, a file called sources.txt must be present. This file will be submitted with your work automatically.

You are to write either of the following statements in the file "sources.txt":

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

Then add your name and your student number as signature

OR:

Write exactly which part of the code of the workshops or the assignment are given to you as help and who gave it to you or which source you received it from.

You need to mention the workshop name or assignment name and also the file name and the parts in which you received the code for help.

Finally add your name and student number as signature.

By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

PROJECT DEVELOPMENT NOTES

- When developing the classes in this project, note that you may add additional member variables, member functions to any class if you find them necessary or helpful.
  Make sure these additional variables and functions are well documented.
- An extra empty module is provided with the project in case you would like to add any helper functions of your own or additional classed to add to your project.
  This module has two files: Utils.h and Utils.cpp.
  Utils.h will be included an all the tester files and Utills.cpp will be added to the compile line of all submissions.
- You may reuse and copy any code your professor provided for your workshops or functions you may have from previous work in this subject or other subjects and place it in the Utils module.

- If you choose not to reuse any of your code, just leave these file empty but have them present when submitting your code.

OVERVIEW OF THE CLASSES TO BE DEVELOPED FOR MILESTONE 1

All the code developed in this project should be under the namespace sdds;

Date Class

A class the encapsulates year, and month and day values for Date stamp, comparison and Date IO purposes.

MenuItem Class

A class that hold a text Item; (an option or title to be displayed) in a menu to be selected by the user. This is a fully private class that is only accessible by Menu (see next class)

Menu Class

A class that has several **MenuItem**s to be displayed so the user can select one of them for an action to be executed in the program

THE DATE CLASS

The Date class was partially implemented by another program that left the company and your responsibly is to reuse the parts she developed to complete the implementation:

The date class incapsulates the following values:

- Year; an integer between the year 1500 till today
- Month, an integer between 1 and 12

- Day, an integer between 1 and the number of days in the month.
- Error code; an integer which holds the code that corresponds to an error that recently happened or ZERO if the date object is valid and ready to be used.
- Current year; an integer that is automatically initialized to the current date of the system for validation purposes when a Date object is instantiated.

The Date module in files Date.h and Date.cpp is well documented and is placed in the project directory.

Study it and learn what each constant, variable and member function does and then using those function and your knowledge of iosteam, cin an cout add the following member functions to the Date class:

```
std::istream& read(std::istream& is = std::cin);
```

This function reads a date from console in following format YYYY/MM/DD as follows:

- Clear the error code by setting it NO_ERROR
- Read the year, the month and the day member variables using istream and ignore a single character after the year and the month values to bypass the Slashes
  > *Note that the separators do not have to be Slash characters "/" but any separator that is not an integer number.*
- Check and see if istream has failed. If it did fail, set the error code to CIN_FAILED and clear the istream.
  If not, validate the values entered.
- Flush the keyboard
- Return the istream object

```
std::ostream& write(std::ostream& os = std::cout)const;
```

If the Date object is in a "bad" state or (it is invalid) print the "**dateStatus()**".
otherwise the function should write the date in the following format using the ostream object:

- Prints the year
- Prints a Slash "/"
- Prints the month in two spaces, padding the left digit with zero, if the month is a single digit number
- Prints a Slash "/"
- Prints the day in two spaces, padding the left digit with zero, if the day is a single digit number
- Makes sure the padding is set back to spaces from zero
- Returns the ostream object.

Operator overloads: (do not use friend)

Overload the following comparison operators to compare two dates.

Use the return value of the **daysSince0001_1_1** member function to compare two dates:

```
bool operator==
bool operator!=
bool operator>=
bool operator<=
bool operator<
bool operator>
```

Use the return value of the **daysSince0001_1_1** member function to overload the

```
int operator-
```

to return the difference in number of days between two dates.
Example:

```
Date
      D1(2019, 12, 02),
      D2(2019, 11, 11);
int days = D1 - D2;
```

"days: in the above code snippet will hold the value 21.

Bool cast overload:

Overload the Boolean cast so if a date is casted to bool, it will return true if the date is valid and false if it is not.

Helper operator overloads:

Overload the following helper operator overloads to have the Date class compatible with cin and cout, input and output operations:

```
operator<<  (for cout)
operator>>  (for cin)
```

DATE FUNCTIONS THAT ARE ALREADY IMPLEMENTED:
Private functions:

```
int daysSince0001_1_1()const; // returns number of days passed since the date 0001/1/1
bool validate();              /* validates the date setting the error code and then
                                 returning the result that is true, if valid, and
                                 false if invalid. */
void errCode(int theErrorCode); // sets the error code
int systemYear()const;         // returns the current system year
bool bad()const;              // return true if
int mdays()const;             // returns the number of days in current month
void setToToday();            // sets the date to the current date (system date)
```
Public Functions and constructors:
```
Date();                            // creates a date with current date
Date(int year, int mon, int day); /* creates a date with assigned values
                                      then validates the date and sets the
                                      error code accordingly */
int errCode()const;        // returns the error code or zero if date is valid
const char* dateStatus()const;  /* returns a string corresponding to the current status
                                   of the date */
int currentYear()const;         // returns the m_CUR_YEAR value;
```

DATE TESTER PROGRAM AND EXECUTION SAMPLE

Write your own tester or use **dateTester.cpp** to make sure your Date Module works correctly.

Compile your **Date** module with either **dateTester.cpp** or **dateSubmissionTester.cpp** for pre-submission test.

You should complete the coding for the Date module in two days.

For execution sample run any of the following commands on matrix:
~fardad.soleimanloo/244/ms1/dateTester
~fardad.soleimanloo/244/ms1/dateSumbissionTester

THE MENU MODULE

Create a module called Menu (in files Menu.cpp and Menu.h) this module will hold both MenuItem and Menu Classes' implementation code.

Forward declare the class Menu in the header file.

THE MENUITEM CLASS

Create a class Called MenuItem. This class is to hold only one C style string of characters for the description in the menu item. The length of the description is unknown.
This class should be fully private (no public members what so ever!).

- Make the "Menu" class a friend of this class (which makes MenuItem class only accessible by the Menu class).

- The description of the MenuItem is only to be set to a value at the moment of instantiation (or initialization) and is not changeable after the MenuItem is created.
  (constructor with DMA)

- If no value is provided for the description at the moment of creation, the MenuItem should be set as empty (with no description).
  (no argument constructor or default argument value and safe empty state)

- A MenuItem object can not be copied from or assigned to another MenuItem object.
  (Copy and Assignment prevention)

- When a MenuItem is casted to "bool" it should return true, if it is not empty and it should return false if it is empty.
  (Cast overload)

- When a MenuItem is casted to "const char*" it should return the address of the description C-string.
  (Cast overload)

- Display the description of the MenuItem using a function that receives an ostream reference argument and returns it when printing is done. If no value is passed as argument to this function it should pass the "cout" object instead by default.

- Make sure there is no memory leak after MenuItem goes out of scope.
  (destructor and DMA)

THE MENU CLASS

Create a class called Menu.
A Menu Object can not be copied or assigned to another Menu Object.
This class has minimum of three member variables.

1- A MenuItem to possibly hold the title of the Menu.
2- An array of MenuItem pointers. The size of this array is set by a constant unsinged integer defined in the Menu header file; called MAX_MENU_ITEMS. Have the MAX_MENU_ITEMS integer initialized to 20.
   This array will keep potential MenuItems added to the Menu. Each individual element of this array will hold the address of a dynamically allocated MenuItem as they are added to the Menu. (See insertion operator overload for Menu)
3- An integer to keep track of how many MenuItem pointers are pointing to allocated memories (obviously the value of this variable is always be between 0 and MAX_MENU_ITEMS).

CONSTRUCTORS, MEMBER VARIABLES AND OPERATOR OVERLOADS

The following are the list of Constructors, member function and operator overloads.
Make them private or public based on your own judgement.
Also make sure those member functions and operator overloads that do not change the Menu class are constant.

- A Menu is always created empty; with no MenuItems, with or without a title. Example:
  Menu A;
  Menu B("Lunch Menu");
- Create a function to display the title of the Menu.
- Create a function to display the entire Menu.
  This function first displays the title (if it is not empty) followed by a ":" and a newline, then it will display all the MenuItems one by one; adding a row number in front of each. The row numbers are printed in two spaces, right justified followed by a "dash" and a "space".

After printing all MenuItems it should print `" 0- Exit"` and new line and `"> "`.
Example:

```
Lunch Menu:
 1- Omelet
 2- Tuna Sandwich
 3- California Rolls
 0- Exit
 >
```

- Overload the insertion operator (operator<<) to add a MenuItem to the Menu.
  This operator receives a C Style string containing the description of the MenuItem and
  return the reference of the Menu object itself.
  To accomplish this, check if a spot for a MenuItem is available in the array of MenuItem
  pointers. If it is, dynamically create a MenuItem out of the description received through
  the function argument and then store the address in the available spot and finally add
  to the number of allocated MenuItem pointers.
  If no spot is available, (that is; if number of allocated MenuItem pointers is equal to
  `MAX_MENU_ITEMS`) this function silently ignores the action.
  At the end, return the reference of the Menu object. Usage example:
  ```
  Menu M;
  M << "Omelet" << "Tuna Sandwich" << "California Rolls";
  ```

- Create a member function called `getSelection`. This function displays the Menu and
  gets the user selection (this function should be completely foolproof)
  The function receives nothing and returns an unsinged integer (That is the user's
  selection).
  After displaying the Menu, ask for an integer and make sure the value of the integer is
  between 0 and the number of the menu items. If the user enters anything incorrect,
  print:
  `"Invalid Selection, try again: "`
  and get the integer again until a valid selection is made.
- Overload `operator~` to do exactly what `getSelection` does.
- Casting the Menu to an integer or an unsigned integer should return the number of
  MenuItems allocated in the MenuItem array of pointers.
- Casting the Menu to "bool" returns true if the Menu has one or more menu items,
  otherwise it returns false.
- Overload the insertion operator to print the title of the Menu using cout.
- Example for last three overloads:
  ```
  Menu M ("Lunch Menu");
  M << "Omelet" << "Tuna Sandwich" << "California Rolls";
  if (M) {
      cout << "The " << m1 << " is not empty and has "
           << unsigned int(M) << " menu items." << endl;
  }
  ```

The above code snippet will print the following:

`The Lunch Menu is not empty and has 3 menu items.`

- Overload the indexing operator to return the const char* cast of the corresponding MenuItem in the array of MenuItem pointers.
  If the index passes the number of MenuItems in the Menu, loop back to the beginning.
  Example:

```
Menu M;
M << "Omelet" << "Tuna Sandwich" << "California Rolls";
cout << M[0] << endl;
cout << M[1] << endl;
cout << M[3] << endl;
```

The above code snippet will print the following:

```
Omelet
Tuna Sandwich
Omelet
```

MENU TESTER PROGRAM AND EXECUTION SAMPLE

Write your own tester or use the tester programs provided to make sure your Menu Module works correctly.

Compile your **Menu** module with either **menuTester.cpp** or **menuSubmissionTester.cpp** for pre-submission testing.

On Matrix compile your cpp files with: **g++ -Wall -std=c++11** command.

You should complete the coding for the Menu module by Monday November 18[th.]

For execution sample run any of the following commands on matrix:
~fardad.soleimanloo/244/ms1/menuTester
~fardad.soleimanloo/244/ms1/ menuSubmissionTester

DUE DATE FOR MILESTONE 1

Suggested due date: Monday November 18th, 2019

Check the exact due dates for your section by adding -due to the end of your submission command:
~profname.proflastname/submit 244/NXX/PRJ/ms1 -due<ENTER>
(use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

SUBMISSION INSTRUCTIONS

To test and demonstrate execution of your program follow the instructions when submitting your code.

If not on matrix already, upload Utils, Date and Menu modules and the tester programs to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/PRJ/ms1<ENTER>

MILESTONE 2, THE READWRITABLE INTERFACE

Create an Interface (a class with pure virtual functions only) called ReadWriteable, in a module called ReadWriteable.

ReadWriteable has two pure virtual member functions:
- read

This function receives and returns references of istream. The receiving argument should be defaulted to the global object cin.

- write

This function receives and returns references of ostream and can not modify the class ReadWriteable. The receiving argument should be defaulted to the global object cout.

Overload helper insertion and extraction operators so any ReadWriteable class can be printed or read like primitive values with cout and cin.

Have the prototypes in ReadWriteable.h (where ReadWriteable class is implemented) and the implementation in ReadWriteable.cpp file.

READWRITEABLE TESTER PROGRAM AND EXECUTION SAMPLE

Write your own tester or use the tester program provided to make sure your ReadWriteable Module works correctly.

Compile your ReadWriteable module with **ReadWriteableTester.cpp** for pre-submission testing.

On Matrix compile your cpp files with: **g++ -Wall -std=c++11** command.

For execution sample run the following command on matrix:
~fardad.soleimanloo/244/ms2/rwtester

DUE DATE FOR MILESTONE 2

Suggested due date: Tuesday November 19th, 2019

Check the exact due dates for your section by adding -due to the end of your submission command:
~profname.proflastname/submit 244/NXX/PRJ/ms2 -due<ENTER>
(use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

SUBMISSION INSTRUCTIONS

To test and demonstrate execution of your program follow the instructions when submitting your code.

If not on matrix already, upload ReadWriteable module and the tester program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/PRJ/ms2<ENTER>

MILESTONE 3, THE PUBRECORD ABSRACT CLASS

Create an abstract class for holding records of publications in a library. Call this class PubRecord and inherit if from ReadWriteable class.

In the header file of PubRecord module, create two global constant integers called SDDS_CONSOLE and SDDS_FILE and set them to two different values (any two integers will suffice).

PubRecord class does not implement the pure virtual methods declared in ReadWriteable class.

Add another pure virtual method called that `recID` to PubRecord that returns a character and does not change the class. Functionality of this function will be explained in the next milestone.

PubRecord has the following member variables:

- A character pointer to holds a dynamic C-style string to hold the name of the publication.
- An integer to hold the shelf number on which the publication is held in the library
- An integer to hold the type of the media on which the PubRecord is to be written. (either SDDS_CONSOLE or SDDS_FILE)

PubRecord has the following protected member functions:

- A function called `name` that receives a constant character pointer to set the name of the publication dynamically.
- Another function called name that returns the name of the publication using a constant character pointer. This overloaded function can not change the state of the class.

Public functions, constructors and operator overloads:

- A no argument (default) constructor that sets the class to an empty state and a copy constructor that safely copies a PubRecord.
- A destructor to make sure there is no memory leak.
- An assignment operator to assign a PubRecord to another PubRecord.
- Two functions called `mediaType` to set and return the type of the media (member variable of PubRecord). Make sure the one returning the type of the media is incapable of changing the state of the class.
- A function called `shelfNo` that returns the integer member variable for the shelf number of the publication. Make sure this function is incapable of changing the state of the class.
- A function called `readShelfNo` that receives a three-digit integer from the console. If an invalid shelf number is entered, print the error message:
  "Invalid Shelf Number, Enter again: "
  and keep asking the user for an integer until a valid shelf number is entered.
- Overload the operator== twice. These two operator overloads both return true or false and they can not change the state of the class. First overload receives a character and compares it to the return value of the `recID` function. If there is a match it will return true, otherwise it will return false.
  Second overload receives a const character pointer for a C-Style string. If this string is a substring of the name of the class, the operator returns true, otherwise it will return false.

  > Hint: use the *strstr* function from the *cstring* library.
  > here is the prototype of the *strstr* function:
  > *const char* strstr(const char* str, const char* substr)*

How it works:
strstr function returns nullptr if the substr argument is not a sub-string of str argument. If it does not return nullptr, it means substr argument is a sub-string of str argument.

- If PubRecord is casted to a bool; it should return true if the PubRecord is not empty This cast overload returns false if PubRecord is empty.

PUBRECORD TESTER PROGRAM AND EXECUTION SAMPLE

Write your own tester or use the tester program provided to make sure your PubRecord Module works correctly.

Compile your PubRecord module with **ms3.cpp** for pre-submission testing.

On Matrix compile your cpp files with: **g++ -Wall -std=c++11** command.

For execution sample run the following command on matrix:
~fardad.soleimanloo/244/ms3/prtester

DUE DATE FOR MILESTONE 3

Suggested due date: Saturday November 23th, 2019

Check the exact due dates for your section by adding -due to the end of your submission command:
~profname.proflastname/submit 244/NXX/PRJ/ms3 -due<ENTER>
(use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

SUBMISSION INSTRUCTIONS

To test and demonstrate execution of your program follow the instructions when submitting your code.

If not on matrix already, upload PubRecord , ReadWriteable, Date, Utils modules and the tester program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/PRJ/ms3<ENTER>

MILESTONE 4, BOOK AND MAGAZINE CLASSES

Before starting milestone 4, apply the following additions to PubRecord class:

```
void shelfNo(int value);
```

Add the above protected method to PubRecord to set the value of m_shelfNo member variable.

```
virtual void checkIn() = 0;
virtual void checkOut() = 0;
```

Add the above public pure virtual methods to enforce "check in" and "check out" actions to derived classes.

MAGAZINE CLASSES

The Magazine class is to encapsulate a periodical publication.

Inherit a class from PubRecord called Magazine. Magazine adds two integer member variables to the class; Volume and Issue number. Volume and Issue number are integer values between 1 and 99, inclusive.

CONSTRUCTORS AND ASSIGNMENT:

A Magazine can be created with no initial data using a no-argument constructor. This constructor invokes the no-argument constructor of the base and initializes the member variables to safe empty states.

A Magazine should be safely copied and assigned to another Magazine with no memory leak.

MEMBER FUNCTIONS.

recID.

Override the recID pure-Virtual function. This function returns the character 'M'. Use this function's return value to tag the Magazine records in the file. (see the Write function).

checkIn and **checkout**

Unlike Books, Magazines are not lent out to the library members therefore these functions only print the following message on the console and then go to newline.
"Magazines cannot be checked in or out!"

read function

Override the read pure-virtual function as follows:

If the mediaType is **SDDS_CONSOLE**, then through interaction with the user get the values of name, volume, issue number and shelf number from user using the following prompts and rules:

Prompt:
`"Magazine Name: "`

Then receive and set the name of the Magazine to a character string up to 40 characters long. If the name is longer than 40 characters print the following error message and repeat your request until the correct value is entered:
`"Magazine name too long, Enter again: "`

Prompt:
`"Volume: "`

Then receive and set the volume number of the Magazine to an integer between 1 and 99 (inclusive). If the value is invalid print the following error message and repeat your request until the correct value is entered:
`"Invalid Volume number, Enter again: "`

Prompt:
`"Issue: "`

Then receive and set the issue number of the Magazine to an integer between 1 and 99 (inclusive). If the value is invalid print the following error message and repeat your request until the correct value is entered:
`"Invalid Issue number, Enter again: "`

Prompt:
`"Shelf Number: "`

Then call the readShelfNo function to set the shelf number.

If the mediaType is **SDDS_FILE**, then without interaction with the user, read name (maximum 40 chars), volume, issue and shelf number from the istream, through a "tab" separated list that ends with a new line.

Example:
`C++ Journal\t2\t3\t111\n`

In this example, the name of the Magazine is "C++ Journal", the volume is "2", the issue number is "3" and the shelf number is "111".

If the istream read all the values **successfully** store the values in the class.

At the end in either console or file mode, return the istream argument.

write function:

Override the write pure-virtual function as follows:

If the mediaType is SDDS_CONSOLE, then print the values to the ostream argument in following format:

- In 41 spaces left justified: print the name
- Print 7 spaces
- In 2 spaces, padded with 0 and right justified: print the volume
- Print "("
- In 2 spaces, padded with 0 and right justified: print the issue number
- Print ")"
- Print the shelf number.

If the media type is SDDS_FILE print the values on the ostream argument in following format:

- Print the recID
- Print the name
- Print the tab character
- Print the volume
- Print the tab character
- Print the issue number
- Print the tab character
- Print the shelf number
- Print a new line.

At the end in either console or file mode, return the ostream argument.

BOOK CLASS

The Book class is to encapsulate any type of Book.

Inherit a class from PubRecord called Book. Book adds the following member variables to the class:

- ISBN number; an integer type large enough to hold a 13-digit number.
- An integer to hold the borrower's membership id. This id is exactly 5 digits long. If a Book is not borrowed and the Book is available in the library, the membership number is be set to 0. (no one borrowed this Book)
- A Date object to indicate the due date for returning a Book to the library after borrowing the Book. The date value is only set and comes to attention if the member id is not 0.
  (A Book with a non zero member id, means that it has been borrowed by a member and is due back at the date stored in the Date object).

CONSTRUCTORS AND ASSIGNMENT:

A Book can be created with no initial data using a no-argument constructor. This constructor invokes the no-argument constructor of the base and initializes the member variables to safe empty states.

A Book should be safely copied and assigned to another Book with no memory leak. When assigned or copied, unlike all other values, the member id of the copy is set to zero (0).

MEMBER FUNCTIONS.

recID.

Override the recID pure-Virtual function. This function returns the character 'B'. Use this function's return value to tag the Book records in the file. (see the Write function).

memberId.

This function returns the member id and can not modify the Book object.

checkIn

This function sets the member id to zero (0) and then prints the Book on the console followed by " checked in!" and goes to new line.

checkOut

This function checks out the Book and lends it to a member by receiving the member id and the date for returning the Book. A Book can not be lent out for more than 30 days.

Here is the process.

First receive and set the member id of the Book to a 5-digit integer. If the value is invalid print the following error message and repeat your request until the correct value is entered:
"Invalid Member ID, Enter again: "

Then get the return date and validate it in a loop until an acceptable return date is entered:
Prompt:
"Enter return date: "

Then read the date and make sure it is a valid date. If not, display the corresponding error (cin Failed, invalid Year and etc) followed by a period "." and go to newline and then go back to the date entry prompt.

If the date is valid, then make sure it is in the future (which means after todays date) if not, show the following error message and to back to the date entry prompt.

"Please enter a future date."

If the date is a future date, make sure is not more than 30 days away. If it is more than 30 days away, print the following error message and go back to the date entry prompt.

"The return date must be earlier than 30 days away from today."

Follow this message with a new line.

Function ends if a **valid** date entry is completed.

read function:

Override the read pure-virtual function as follows:

If the mediaType is **SDDS_CONSOLE**, first set the member id to zero (0) then through interaction with the user get the values of name, ISBN, and shelf number from user using the following prompts and rules:

Prompt:
`"Book Name: "`

Then receive and set the name of the Book to a character string up to 40 characters long. If the name is longer than 40 characters print the following error message and repeat your request until the correct value is entered:
`"Book name too long, Enter again: "`

Prompt:
`"ISBN: "`
Then receive and set the ISBN number of the Book to a 13-digit integer. If the value is invalid print the following error message and repeat your request until the correct value is entered:
`"Invalid ISBN, Enter again: "`

Prompt:
`"Shelf Number: "`

Then call the readShelfNo function to set the shelf number.

If the mediaType is **SDDS_FILE**, then without interaction with the user, read name (maximum 40 chars), ISBN, shelf number and member id using the istream, through a "tab" separated list. Then check the member id, if it is zero, ignore one character (the newline) and set the member variables to the values read and return the istream argument.
If the member id is not zero, then ignore one character(the tab) and read the date using the istream argument and like before, set the member variables to the values read and return the istream argument.
Example for a record with 0 for member id:
`C++ 244\t9876543212345\t333\t0\n`
  (A book with the title "C++ 244", ISBN number: 9876543212345 and shelf number 333)
Example for a record with a non-zero member id:
`C 144\t1234567890987\t222\t12345\t2019/12/24\n`
  (A Book with the title C 144, ISBN number: 1234567890987, shelf number: 222 that is lent to member number 12345 and is due to be returns on 2019/12/24)

At the end in either console or file mode, return the istream argument.

`write` function:

Override the write pure-virtual function as follows:

If the mediaType is **SDDS_CONSOLE**, then print the values on the ostream argument in following format:

- In 41 spaces left justified: print the name
- Print the ISBN number
- Print 2 spaces
- Print the shelf Number
  if member id is not zero
    - o Print one space
    - o Print the member id
    - o Print 3 spaces
    - o Print the Date.

If the media type is **SDDS_FILE** print the values on the ostream argument in following format:

- Print the recID
- Print the name
- Print the tab character
- Print the ISBN
- Print the tab character
- Print the shelf Number
- Print the tab character
- Print the member id
- If member id is not zero
    - o Print the tab character
    - o Print the date
- Print a new line.

At the end in either console or file mode, return the ostream argument.

MAGAZINE AND BOOK TESTER PROGRAM AND EXECUTION SAMPLE

Write your own tester or use the tester program provided to make sure your Magazien and Book Modules works correctly.

Compile your Magazine and Book module with **ms4.cpp** for pre-submission testing.

(V.09)Compile your Magazine and Book module with **ms4valid.cpp** for value entry validation tests.

On Matrix compile your cpp files with: **g++ -Wall -std=c++11** command.

For execution sample run the following command on matrix:
~fardad.soleimanloo/244/ms4/mbtester

DUE DATE FOR MILESTONE 4

Suggested due date: Saturday November 30th, 2019

Check the exact due dates for your section by adding -due to the end of your submission command:
~profname.proflastname/submit 244/NXX/PRJ/ms4 -due<ENTER>
(use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.)

SUBMISSION INSTRUCTIONS

To test and demonstrate execution of your program follow the instructions when submitting your code.

If not on matrix already, upload all your modules and the tester program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/PRJ/ms4<ENTER>

PROJECT MARKING, MILESTONE 4:

If you have submitted your milestone 4 successfully, you may receive maximum of 85% of the project.

Complete the following final Validation Submission to get 100% for your project.

FINAL VALIDATION SUBMISSION

Before doing your final validation submission correct the typo in Book.cpp in function checkOut():

Correct: "Entere Member id: " message to "Enter Member id: "

After correcting the spelling mistake in the message of the checkout function submit your milestone 4 as follows:

Run the following script from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

`~profname.proflastname/submit 244/NXX/PRJ/val`<ENTER>

This command first copies the following files to your ms4 directory:
libdata.dat, library.cpp, PubApp.h and **PubApp.o**

And then compiles and runs a library application using the tools you have developed in the 4 milestones.

Test your application using the data provided in the validation execution output section.

If your code does not pass the test, to debug your code in Visual studio:

Copy **libdata.dat, library.cpp, PubApp.h** and **PubApp.obj** from the PrjVal directory of the Project repository into your ms4 directory. Then in visual studio/solution explorer, add the **library.cpp** file to your source files, add the **PubApp.h** file to your header files and the **PubApp.obj** file to your recourse files and remove the **ms4.cpp** file.

Make sure the debug setting is set to x86:
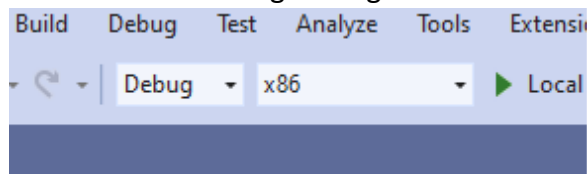
Now you can compile and debug your program.

On Linux, add the two files: `library.cpp` and **PubApp.o** to your compile command line.

DUE DATE FOR PROJECT VALIDATION

Check the exact due dates for your section by adding -due to the end of your submission command:
`~profname.proflastname/submit 244/NXX/PRJ/val -due`<ENTER>
(use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.)

PROJECT VALIDATION EXECUTION OUTPUT.

For execution sample run the following command on matrix:
~fardad.soleimanloo/244/val/app

User Entries are in RED


```
My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 4
Search for name: Can
Searching....

Row  Title                                ISBN or V(I)  Shlf Lent to Ret Date
--------------------------------------------- ------------- --- -----   ----------
  1- Canadian Living                         14(01)  155
  2- Canadian Living                         14(10)  155
  3- Canadian Living                         14(16)  155
  4- Literary Review of Canada              05(04)  132
  5- Literary Review of Canada              05(10)  132
  6- Literary Review of Canada              05(12)  132
  7- Literary Review of Canada              05(16)  132
  8- Literary Review of Canada              05(18)  132
  9- Canadian Geographic                    03(02)  122
  0- Exit
Select: 0

My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 4
Search for name: Th
Searching....

Row  Title                                ISBN or V(I)  Shlf Lent to Ret Date
--------------------------------------------- ------------- --- -----   ----------
  1- The Hockey News                         10(01)  111
  2- The Hockey News                         10(04)  111
  3- The Hockey News                         10(07)  111
  4- The Hockey News                         10(10)  111
  5- The Fun Family                    9781603093446  244
  6- That Salty Air                    9781603090056  244
  0- Exit
Select: 0

My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
```

```
> 2
Checking Out:
Search for name: Fun
Searching....

Row  Title                                     ISBN or V(I)  Shlf Lent to Ret Date
---------------------------------------------- ------------  --- -----   ----------
  1- The Fun Family                            9781603093446  244
  0- Exit
Select: 1

Checking out:
The Fun Family                        9781603093446  244
Enter Member id: 111111
Invalid Member ID, Enter again: 1111
Invalid Member ID, Enter again: 11111
Enter return date: 2019/10/10
Please enter a future date.
Enter return date: 2019/12/12

My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 4
Search for name: Th
Searching....

Row  Title                                     ISBN or V(I)  Shlf Lent to Ret Date
---------------------------------------------- ------------  --- -----   ----------
  1- The Hockey News                                10(01)  111
  2- The Hockey News                                10(04)  111
  3- The Hockey News                                10(07)  111
  4- The Hockey News                                10(10)  111
  5- The Fun Family                            9781603093446  244 11111   2019/12/12
  6- That Salty Air                            9781603090056  244
  0- Exit
Select: 0

My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 3
Checking In:
Search for name: Fun
Searching....

Row  Title                                     ISBN or V(I)  Shlf Lent to Ret Date
---------------------------------------------- ------------  --- -----   ----------
  1- The Fun Family                            9781603093446  244 11111   2019/12/12
  0- Exit
Select: 1

Checking in:
```

```
The Fun Family                          9781603093446  244 11111   2019/12/12
The Fun Family                          9781603093446  244 checked in!

My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 1
What type of Publication?:
 1- Magazine
 2- Book
 0- Exit
> 1
Magazine Name: The Guardian
Volume: 10
Issue: 6
Shelf Number: 123
Publication Added.

My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 1
What type of Publication?:
 1- Magazine
 2- Book
 0- Exit
> 2
Book Name: The Surrogates
ISBN: 9781891883087
Shelf Number: 200
Publication Added.

My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 4
Search for name: The
Searching....

Row  Title                              ISBN or V(I)  Shlf Lent to Ret Date
---------------------------------------- ------------- --- -----   ----------
   1- The Hockey News                         10(01)  111
   2- The Hockey News                         10(04)  111
   3- The Hockey News                         10(07)  111
   4- The Hockey News                         10(10)  111
   5- The Fun Family                   9781603093446  244
   6- The Guardian                            10(06)  123
   7- The Surrogates                   9781891883087  200
   0- Exit
Select: 0
```

```
My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 1
What type of Publication?:
 1- Magazine
 2- Book
 0- Exit
> 2
Book Name: aaasssdddfffaaasssdddfffaaasssdddfffaaasss
Book name too long, Enter again: abc
ISBN: 12345678909
Invalid ISBN, Enter again: 12345678909876
Invalid ISBN, Enter again: 1234567890987
Shelf Number: 11
Invalid Shelf Number, Enter again: 1111
Invalid Shelf Number, Enter again: 111
Publication Added.

My Little Library:
 1- Add Publication
 2- Check Out
 3- Check In
 4- Search
 0- Exit
> 0
Do you really want to exit the program?:
 1- No, Cancel!
 0- Exit
> 0
Save and overwrite the old data?:
 1- Yes, overwrite the data file with the current data
 0- Exit
> 1
33 Item(s) saved.
```

PROJECT BONUS TASK

For 10% bonus mark, create your own library application that uses the tools you have written to perform a similar task as the validation program.

Use **libdata.dat** as your data file and write your application in a module called PubApp (**PubApp.h** and **PubApp.cpp**) and your main function in **library.cpp.**

You are free to write your application in anyway you want.

After completing your code, run and capture the execution sample of your program using the following submit command.

Make sure you demonstrate all the capabilities of your application.

PROJECT BOUS SUBMISSION

`~profname.proflastname/submit 244/`**`NXX`**`/PRJ/app` <ENTER>

(This command will not validate your execution and submits the demonstration to your professor)