Fall Semester 2019

Library Publication Storage and Retrieval Application

V0.9(Milestone 3)

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 18th.

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 header file of `PubRecord` module Create two global constant integers called `SDDS_CONSOLE` and `SDDS_FILE` and set them to two different values.

`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 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 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 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>