Assignment 2 - 100 points

Part 2

Assignment Overview

In Part 2 of this assignment, you will add some functionality to the `BookStore` class and add some logic to your `main()` routine to test that functionality.

Purpose

This part of the assignment introduces the insertion sort and binary search algorithms. It also introduces the use of makefiles for compiling and linking your programs.

Set Up

1. In your Assignment 2 directory, make a symbolic link to the data file for this part of the assignment:

   ```
   ln -s /home/turing/t90kjm1/CS241/Data/Fall2016/Assign2/orders.txt
   ```

2. In this assignment, you will be creating several source code and header files, as described below. You can create each of these files separately using the `nano` editor, just as you did on Assignment 1.

3. Once you've written a makefile for your program, you can compile and link it by simply typing:

   ```
   make
   ```

4. Running the executable file is unchanged from Part 1.

5. To remove the executable and object code files for your program (conserving disk space), you can type:

   ```
   make clean
   ```

Program

For this part of the assignment, you'll need to write one new file and then modify two of the files you wrote for Part 1.

Step 1: Write a makefile

The file named `makefile` tells the `make` utility how to build the final executable file from your collection of C++ source code and header files. The makefile for this assignment is given in its entirety below. Makefiles for future assignments will follow this basic pattern.

IMPORTANT NOTE: The commands that appear in the makefile below **MUST** be indented as shown. Furthermore, the indentation **MUST** be done using a tab character, not spaces. If you don't indent your makefile commands, or indent using spaces, your makefile **WILL NOT WORK**. Copying and pasting the text below into a text editor may result in the tabs being replaced with spaces. If so, you will need to manually replace the spaces with a tab instead.

```
#
# PROGRAM:    assign2
# PROGRAMMER: your name
# LOGON ID:   your z-id
# DATE DUE:   due date of program
#

# Compiler variables
CCFLAGS = -Wall -std=c++11
```

```
# Rule to link object code files to create executable file
assign2: assign2.o Book.o BookStore.o
        g++ $(CCFLAGS) -o assign2 assign2.o Book.o BookStore.o

# Rules to compile source code files to object code
assign2.o: assign2.cpp BookStore.h
        g++ $(CCFLAGS) -c assign2.cpp

Book.o: Book.cpp Book.h
        g++ $(CCFLAGS) -c Book.cpp

BookStore.o: BookStore.cpp BookStore.h
        g++ $(CCFLAGS) -c BookStore.cpp

BookStore.h: Book.h

# Pseudo-target to remove object code and executable files
clean:
        -rm *.o assign2
```

Once you've written the file `makefile`, try using the `make` utility to compile and link your program.

Step 2: Add the following methods to the `BookStore` class

- `sortByISBN()` - This method takes no parameters and returns nothing.

  This method should sort the array of `Book` objects in ascending order by ISBN using the insertion sort algorithm.

  The sort code linked to above sorts an array of integers called `numbers` of size `size`. You will need to make a substantial number of changes to that code to make it work in this program:

  1. This will be a method, not a function. Change the parameters for the method to those described above.

  2. In the method body, change the data type of `bucket` to `Book`. This temporary storage will be used to swap elements of the array of `Book` objects.

  3. In the method body, change any occurrence of `numbers` to the name of your array of `Book` objects and `size` to `numBooks` (or whatever you called the data member that tracks the number of valid `Book` objects stored in the array).

  4. The comparison of `bookArray[j-1]` and `bucket` will need to use the C string library function `strcmp()` to perform the comparison. Also, you'll need to use the `getISBN()` method to access the ISBN data member within each `Book` object. The final version of the inner `for` loop should look something like this:

     ```
     for (j = i; (j > 0) && (strcmp(bookArray[j-1].getISBN(), bucket.getISBN()) > 0); j--)
        ...
     ```

  5. It is legal to assign one `Book` object to another; you don't need to write code to copy individual data members.

  Add a call to the `sortByISBN()` method to the end of the alternate constructor you wrote for the `BookStore` class. This will sort the array of `Book` objects that were read in from the input file.

- `searchForISBN()` - This method should take one parameter: a character array containing the ISBN of the `Book` to search for (`searchISBN`). The method should return an integer.

  The logic for this method is a variation of the binary search of a sorted list strategy.

  ```
  int low = 0;
  int high = number of valid Book objects in the array - 1;
  int mid;

  while (low <= high)
     {
     mid = (low + high) / 2;

     if (searchISBN is equal to ISBN data member of bookArray[mid])
        return mid;

     if (searchISBN is less than ISBN data member of bookArray[mid])
        high = mid - 1;
  ```

```
      else
         low = mid + 1;
      }

   return -1;
```

As usual, you'll need to use `strcmp()` to perform the comparison of ISBNs.

- `processOrders()` - This method should take one parameter: a pointer to a constant character (that points to an array of characters containing the name of a file of order records). The method should return nothing.

  This method will read a series of order records, each containing an order number, an ISBN, and an order quantity. This data is not in binary form, so you can not use the same technique to read it that was used for the book data in Part 1. Use the >> operator to read the individual data items into three separate variables.

  For each order record, the method should search for the ISBN and, if found, should call the `fulfillOrder()` method for that book, passing it the order quantity. If an ISBN is not found, an error message should be printed.

  Pseudocode for the method logic is given below.

```
   ifstream inFile;
   char orderNumber[7]
   char isbn[11];
   int orderQuantity;
   int numShipped;

   Open inFile using the file name passed in as a parameter
   Check for successful open

   Read orderNumber from inFile
   while (not end of file)
      {
      Read isbn from inFile
      Read orderQuantity from inFile

      int index = searchForISBN(isbn);
      if (index == -1)
         Print an error message for this order
      else
         {
         Call the fullfillOrder() method for the Book object at element index of the array of Book objects.
           Pass the orderQuantity to the method and store the returned value in numShipped
         Print the results for this order
         }

      Read orderNumber from inFile
      }

   Close input file
```

Step 3: Add two method calls to the main program

The `main()` routine logic will now look like this:

- Create a `BookStore` object using the alternate constructor you wrote. Pass the filename string `"bookdata"` as an argument to the constructor.
- Call the `print()` method for the `BookStore` object.
- Call the `processOrders()` method for the `BookStore` object. Pass the filename string `"orders.txt"` as an argument to the method.
- Call the `print()` method for the `BookStore` object.

Other Points

- As always, programs that do not compile on `turing/hopper` automatically receive 0 points.

- Make sure to document your program according to the standards listed in the Course Notes book. In particular, each function or method should have a documentation box describing the purpose of the function, the input parameters, and the return value (if

any). There should also be a documentation box for the program as a whole.

- Submit the final version of your program using the electronic submission guidelines posted on the course web site and described in class.