Lab 9 – C++ Classes and Vectors
CPSC 1021- Fall 19

Due Date: 10 November, 2019 @ Midnight

Introduction

In today's lab, we will be implementing functions to maintain a list of books. We will also continue to practice with, vectors, I/O manipulation and multiple classes.

Lab Objectives

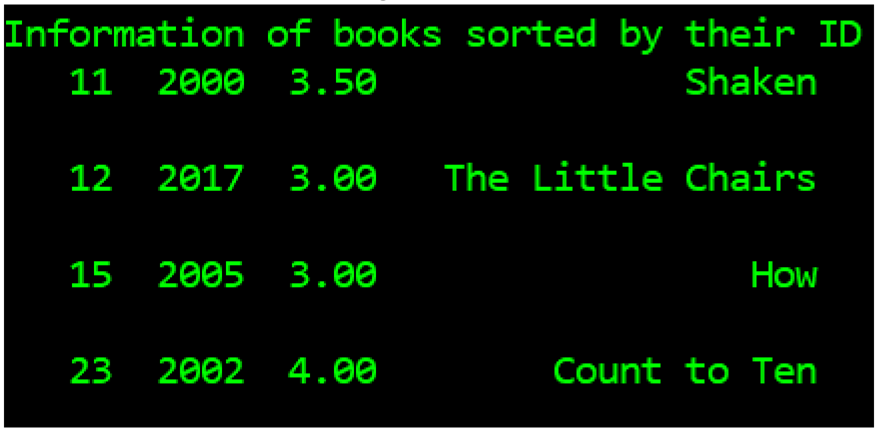
- C++ File I/O + Manipulators
- Working with multiple class with has-a relation
- Learning helpful functions provided for vector manipulation
- Learn how to format output

Lab Instructions

You will be provided with header files, Book.h, BookList.h and sample input file. You have to create 3 files, driver.cpp, Book.cpp, and BookList.cpp. Your program **will read the input file name through command line arguments**. **You can only use C++ style I/O manipulators. You must check if both files opened correctly and the appropriate number of arguments were passed to the command line. Output will be written to stdout using the format shown in this document.**

Book Class

You will be provided with the **book.h** file. You must create a book.cpp file and implement all the functions defined in book.hpp. A book has several attributes, title, id, publication year, rating. Book class has several constructors that you need to implement. If you want to print information for list of books, the following screenshot shows how you should format all the attributes.



```
Information of books sorted by their ID
   11   2000   3.50                  Shaken

   12   2017   3.00   The Little Chairs

   15   2005   3.00                     How

   23   2002   4.00           Count to Ten
```

Since rating is stored as a double, to print it with two decimal point, you can use **std::fixed, std::showpoint, std::setprecision(int)** provided in “iomanip” header. You can refer to lecture slides to see the usage. The **print** function should use a **stringstream** to build and return a string for each book that.

BookList Class

This class contains a vector of raw pointers to Book objects. It has several member functions which you will need to implement. In the constructor for the Book class, the initial size of the vector is passed as a parameter, you should use this parameter to **reserve** memory for the vector. Size is determined by the number of books given in the input text file. Number of books would be counted using the function int howMany(ifstream& input). Basically you go through the file until you reach the end and keep track how many books you have passed.

Keep in mind, you will need to read the file again to store the necessary information for book in vectors. This will require you to reset the pointer back to the beginning of the file. This can be done by using the following two lines of code:

```
input.clear();
input.seekg(0, std::ios::beg);
```

For each book read from the input file, you will need to call the addBook function. This function expects a pointer to dynamically allocated memory.

Fun Fact: You can call the **new** function as a parameter where dynamically allocated memory is expected. The allocated Book object will then be added to the of books. vector<Book*> books .

If you have a class called C, to dynamically create an object you call “ new C()” which uses the default constructor of the class C. This could be done in a function by doing the following: someFunction(new C()), where someFunction is expecting dynamically allocated memory.

- addBook function first searches the vector to see if a book with the same ID value already exists in the vector. If it does, then it doesn’t add the book to the vector and returns false. If not, then uses emplace_back to put the book into vector and returns true.
- Search function searches the vector for a book with the given book ID. If the book is found, it is returned; if it is not found, nullptr is returned. You can use brute force to search each element or take advantage of the function find_if in the **algorithm** library.

Don’t forget to include the algorithm header. An example of using the function is given below for a vector called myVec:

```
auto result=  std::find_if(myVec.begin(),myVec.end(),
                 [value_to_match](const myVec* e)
                  { if(e==nullptr)
                   { return false; }
                   else
                  {    return e->getValue()==value_to_match; ) });
```

The above example uses a lambda function which you may not be familiar with. Lambda functions were added with C++11, so you must use -std=c++11 when compiling. Below are several websites you can visit to learn more about lambda functions.

https://www.drdobbs.com/cpp/lambdas-in-c11/240168241

https://www.geeksforgeeks.org/lambda-expression-in-c/

https://www.google.com/search?client=firefox-b-e&q=lambda+function+C%2B%2B

The one above is a link to a PDF.

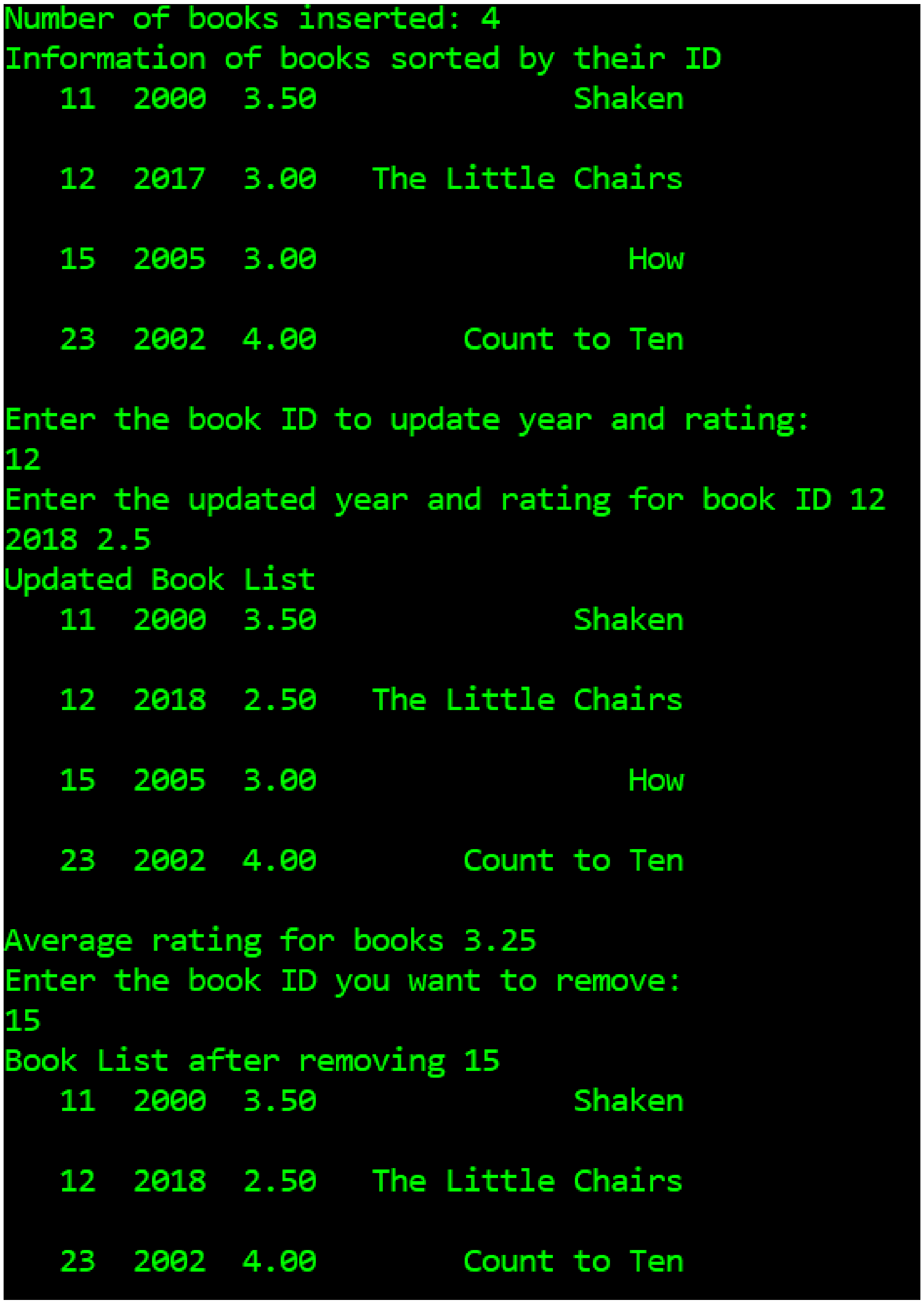
https://www.softwaretestinghelp.com/lambdas-in-cpp/

Find_if returns the first element in the vector that satisfies specific criteria, which in our case would be to match the bookID. Returned value is an iterator to the element found. It is stored in variable of type auto or you can use vector<Book*>::iterator it.

- updateBook function searches for a book with given ID and updates the year and rating for that book. Value to be updated are taken through prompting user.
- calculateAverageRating function calculates the average rating for all books and returns the average
- print function prints out the vector of books to standard output. The books are printed in order of book ID (from smallest to largest), one per line. That means you have to call sort in algorithm library to sort the vector first. You will need to write “compare” function to sort vector of pointers to books.
- Destructor for BookList needs to delete pointers first before it clears the vector itself. If vector is cleared first, you loose track of the allocated memory leak and leak occurs.
- removeBook function searches the list for a book with the given bookID. If the book is found, the book is removed from the list. You can use functions std::erase and

std::remove. Make sure to call delete on the pointer for the book object you want to delete before removing it from the vector to avoid memory leak or dangling pointers.

Sample output looks like the following:



```
Number of books inserted: 4
Information of books sorted by their ID
   11  2000  3.50              Shaken

   12  2017  3.00   The Little Chairs

   15  2005  3.00                 How

   23  2002  4.00        Count to Ten

Enter the book ID to update year and rating:
12
Enter the updated year and rating for book ID 12
2018 2.5
Updated Book List
   11  2000  3.50              Shaken

   12  2018  2.50   The Little Chairs

   15  2005  3.00                 How

   23  2002  4.00        Count to Ten

Average rating for books 3.25
Enter the book ID you want to remove:
15
Book List after removing 15
   11  2000  3.50              Shaken

   12  2018  2.50   The Little Chairs

   23  2002  4.00        Count to Ten
```

What to turn in

- **driver.cpp, Book.cpp, Book.h, BookList.cpp, BookList.h**

Compile and Execute

Use g++ to compile your code as follows **and include the C++11 standard!**:

```
g++ -std=c++11 -Wall -g driver.cpp Book.cpp BookList.cpp -o driver
```

Execute the program

```
./driver <input file name>
```

FORMATTING:

1. Your program should be well documented
   2. Each file should include a header:
   3. Your program should consist of proper and consistent indention
   4. No lines of code should be more than 80 characters

```
// Sample Header
/*******************
  your name
  username
  Lab 1
  Lab Section:
  Name of TA
*******************/
```

5 – 10 points will be deducted for each of the above formatting infractions.

Submission Instructions

- Test your program on the School of Computing server prior to submitting.
- Use the tar utility to tar.gz all source files. **Do not tar an entire directory! When I untar your archive, I should see all the files you included, not a top-level directory! Failure to correctly tar may result in up to a 25-point penalty!**

  EX. tar –czvf yfeaste-lab9.tar.gz *.cpp *.h

- Name your tarred file **<username>-lab<#>.tar.gz** (ex. yfeaste-lab9.tar.gz)
- Use handin (http://handin.cs.clemson.edu) to submit your archive