

# Homework 7: Part II

\*\*The work you do in this assignment will be used as part of project 2

**Due Saturday, October 27<sup>th</sup>, 11:55 pm**

**+5% bonus if submitted by Thursday, October 25<sup>th</sup> 11:55 pm,**

**+2% bonus if submitted by Friday, October 26<sup>th</sup> 11:55 pm**

This assignment is due **October 27<sup>th</sup>, at 11:55 pm**

- **All components (Cloud9 workspace, moodle quiz attempts, and zip file) must be completed and submitted by Saturday, October 27<sup>th</sup>, by 11:55 pm for your homework to receive points.**
- Complete submissions (Cloud9 workspace, moodle Ingenious attempts, and zip file) before **Thursday, October 25<sup>th</sup> 11:55 pm** will receive a 5% bonus, and complete submissions before **Friday, October 26<sup>th</sup> 11:55 pm** will receive a 2% bonus.

---

Objectives:

- Practice implementing classes
  - Develop proper techniques for object-oriented programming
  - Manipulate arrays of objects
- 

## Problem Set

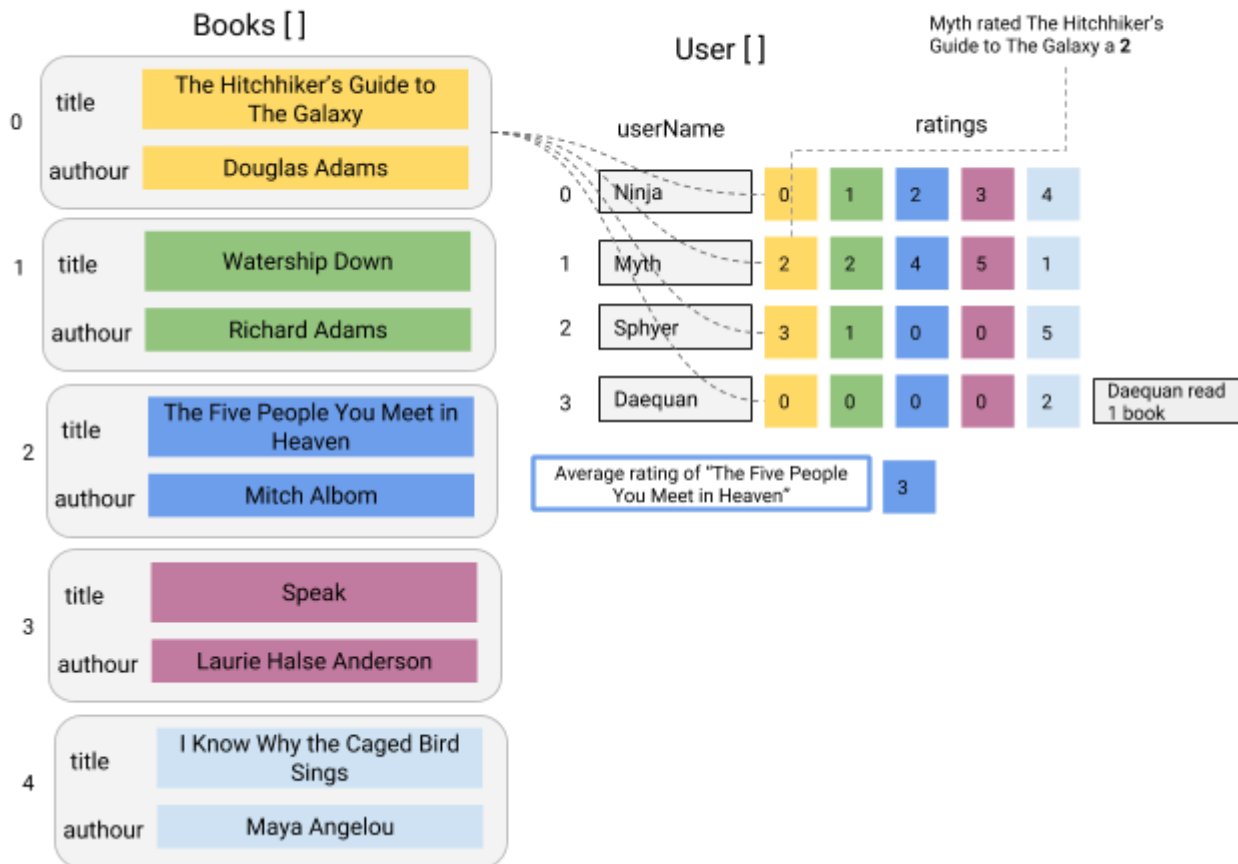
In part 2 you will be adapting the code you wrote in part 1 to be more object oriented. You'll realize that organizing your data in objects not only makes your code more sleek, but also makes it more dynamic. Classes are perhaps the most fundamental data structure in C++ and their use is ubiquitous in the "real world." In this assignment you will become more familiar with the relationship between classes and objects while observing some of the benefits of this data abstraction.

When creating classes, it is good programming practice to separate interface and implementation. This means defining your class in an `.h` file and implementing it in a `.cpp` file of the same name. Ultimately you will hand in five separate files, two for each class, and one driver file that will include a main function and your solutions from problems 3, 4 and 5.

## Specifications

- Create two classes, Book and User. Define the two classes in header files and implement the classes in cpp files.
- In `main()` create arrays of Book and User objects, size 200 (Assume that we don't have more than 200 books and more than 200 users)
- Student should have five files (Book.h, Book.cpp, User.h, User.cpp, Hmwk7.cpp)
- The name of each member function should be exactly the as specified. If you modify the function names your solution will not pass the autograder.

### Visualization of various elements in HW7



## Problem 1

**\*\*You should have separate files for class definition and implementation: Book.h and Book.cpp**

Create a class `Book`, with separate interface and implementation, comprised of the following attributes:

Data members (private):	
string: <code>title</code>	
string: <code>author</code>	
Member functions (public):	
Default constructor	Sets both <code>title</code> and <code>author</code> to empty strings
Parameterized constructor	Takes two strings for initializing <code>title</code> and <code>author</code> , in this order
<code>getTitle()</code>	Returns <code>title</code> as a string
<code>setTitle(string)</code>	(void) Assigns <code>title</code> the value of the input string
<code>getAuthor()</code>	Returns <code>author</code> as a string
<code>setAuthor(string)</code>	(void) Assigns <code>author</code> the value of the input string

It is advisable to write your own test cases for each class. Test your class in Cloud9 before submitting to the autograder, as the autograder has a **submission limit of 20 tries**.

## Problem 2

**\*\*You should have separate files for class definition and implementation: User.h and User.cpp**

Create a class `User`, with separate interface and implementation, comprised of the following attributes:

Data members (private):	
string: <code>username</code>	
int array: <code>ratings</code>	Number of elements should be <code>size</code>
int: <code>numRatings</code>	Number of books in the database

Int: <code>size</code>	The capacity of the <code>ratings</code> array (200). Constant
<b>Member functions (public):</b>	
Default constructor	Sets <code>username</code> to an empty string, <code>numRatings</code> to 0, <code>size</code> to 200, and all the elements of <code>ratings</code> array to the value -1
Parameterized constructor	Takes a string, an array of integers, and two integers for initializing <code>username</code> , <code>ratings</code> , <code>numRatings</code> , and <code>size</code> , respectively
<code>getUsername()</code>	Returns <code>username</code>
<code>setUsername(string)</code>	(void) Assigns <code>username</code> the value of the input string
<code>getRatingAt(int)</code>	Parameter: <code>int index</code> . Returns the rating stored at the specified index. If <code>index</code> is larger than the size of the <code>ratings</code> array, returns -1.
<code>setRatingAt(int,int)</code>	Parameters: <code>int index</code> , <code>int value</code> . Sets the rating to value at the specified index, if <code>index</code> is within the bounds of the array and <code>value</code> is between 0 and 5. Returns a boolean, <code>true</code> if the rating is successfully updated and <code>false</code> otherwise.
<code>getNumRatings()</code>	Returns <code>numRatings</code>
<code>setNumRatings(int)</code>	(void) Assigns <code>numRatings</code> the value of the input <code>int</code>
<code>getSize()</code>	Returns <code>size</code>

It is advisable to write your own test cases for each class. Test your class in Cloud9 before submitting to the autograder, as the autograder has a **submission limit of 20 tries**.

### Problem 3

Write a function `readBooks` that populates an array of `Book` objects with the title and author data found in the file `books.txt`. This function should:

- Accept four input arguments in this order:
  - `string`: the name of the file to be read
  - array of `Book` objects: books data

- int: the number of `Book` objects currently stored in the array of `Book` objects
  - int: capacity of the library system *[assume a max of 200 books]*
- Use `ifstream`, `split()`, and `getline` to read and parse data from the file.
- For each line in the file:
  - instantiate a `Book` object,
  - fill in the `author` and `title` data members, and
  - append the object to your array of `Book` objects.
- Return the total number of books in the system, as an integer.
- If the file cannot be opened, return -1

**Important:** when testing your `readBooks` function, make sure it supports multiple calls in a row. For example, you should be able to call the function to read the file `books1.txt`, and then call the function again to read the file `book2.txt`. The result should be an array of `Book` objects, with the books from the first file, followed by the books from the second file.

## Problem 4

Write a function `readRatings` that will populate an array of `User` objects with the name and rating values from the file `ratings.txt`. Each username represented in `ratings.txt` is followed by list of integers--ratings of each book in `books.txt`.

Rating	Meaning
0	Did not read
1	Hell No - hate it!!
2	Don't like it.
3	Meh - neither hot nor cold
4	Liked it!
5	Mind Blown - Loved it!

This function should:

- Accept four arguments in this order:
  - string: the name of the file to be read
  - array of `User` objects: user data
  - int: number of users currently stored in the array of `User` object
  - int: the capacity of the user array *[assume a max of 200 users]*
- Use `ifstream`, `split()`, and `getline` to read and parse data from the file.

- For each line in the file
  - instantiate a `User` object,
  - fill in the `username` data member,
  - set the `size` data member equal to the capacity parameter
  - populate the ratings array with the data in the file, and fill the rest of the values in the array with the value `-1`
  - store a count for the number of ratings that are not `-1` in the `numRatings` data member, and
  - append the object to your array of `User` objects.
- Print the username of each user as they are added to the system:
 

```
cout << user.getUsername() << "... " << endl;
```
- Return the total number of users in the system, as an integer.
- If the file cannot be opened, return `-1`

**Important:** when testing your `readRatings` function, make sure it supports multiple calls in a row. For example, you should be able to call the function to read the file *ratings1.txt*, and then call the function again to read the file *ratings2.txt*. The result should be an array of `User` objects, with the users from the first file, followed by the users from the second file.

Expected output:
cynthia... diane... joan... barbara... (etc.)

## Problem 5

It will be useful to display the contents of your library. Next, make a function `printAllBooks` that meets the following criteria:

- Accept three arguments in this order:
  - array of `Book` objects: books data
  - int: the number of books currently stored in the arrays of `Books` objects
- This function does not return anything
- If the number of books is 0, print `"No books are stored"`

- Otherwise, print “Here is a list of books”, followed by each book in the following format

```
<book title > by <book author>
```

## Driver function

Menu functionality is the same as in Homework 6. Please use the same menu function provided in Homework 6. The output of your Homework 7 should match the output from Homework 6, even if you are modifying the functions at Problems 3, 4 and 5.

For Homework 7, we will be testing only options 1, 2, 3 and 6. Options 4 and 5 are not required in this homework. However, they will be required in the following homework.

**Note: the main function, menu function, and the function definitions for Problems 3, 4 and 5 should be included in Hmwk7.cpp.**

**At the top of the Hmwk7.cpp file, don't forget to include the header files for the two classes: Book and User**

This is a menu-driven program, where the user is continually being offered six options, until they opt to quit.

### 1. (graded) Initialize library

- Prompt the user for a file name.
- Pass the file name to your `readBooks` function.
- Print the total number of books in the database in the following format:

```
Total books in the database: <numberOfBooks>
```

- If no books are saved to the database due to wrong file name, then print the following message:

```
No books saved to the database
```

### 2. (graded) Initialize user catalog

- Prompt the user for a file name.
- Pass the file name to your `readRatings` function
- Print the total number of users in the database in the following format:

Total users in the database: <numberOfUsers>

- If no books are saved to the database due to wrong file name, then print the following message:

No users saved to the database

3. (graded) Display library contents

- Call your `printAllBooks` function.

4. (not graded) Get number of books reviewed by a user

- Prompt the user for a username.
- Pass the username to your `getUserReadCount` function
- If the user exists in the system, print the result in the following format:

<name> rated <numBookRead> books

5. (not graded) Get average rating for a title

- Prompt the user for a title.
- Pass the title to your `calcAvgRating` function
- If the title exists in the database, print the result in the following format:

The average rating for <bookTitle> is <value>

Note: <value> is a double with 2 decimal points.

6. (graded) Quit

- Print "good bye!" before exiting

## IMPORTANT: How to compile multiple .cpp files and .h files

In this homework, it's required to write multiple files (.h and .cpp files) and test them before submitting them to the Ingenious autograder. You need to compile and execute your code **via command line**. This means you need to type commands in a **bash** window, instead of pushing the *Run* button, like before.



Make sure you first change directory to the folder where your solution files are stored. Here, the folder is `hmkw7`. The command is `cd`, followed by the folder's name:

```
cd hmkw7/
```

When compiling in command line you need to specify all the `.cpp` files in your project. One example of the command for compiling the codes is:

```
g++ -std=c++11 file1.cpp file2.cpp main.cpp
```

The compiling command results in the creation of an executable file. If you did not specify a name for the executable file (like in the example above), the executable will be named `a.out`. To execute the file, use the following command:

```
./a.out
```

You can add `-o` flag to specify the name of the executable file:

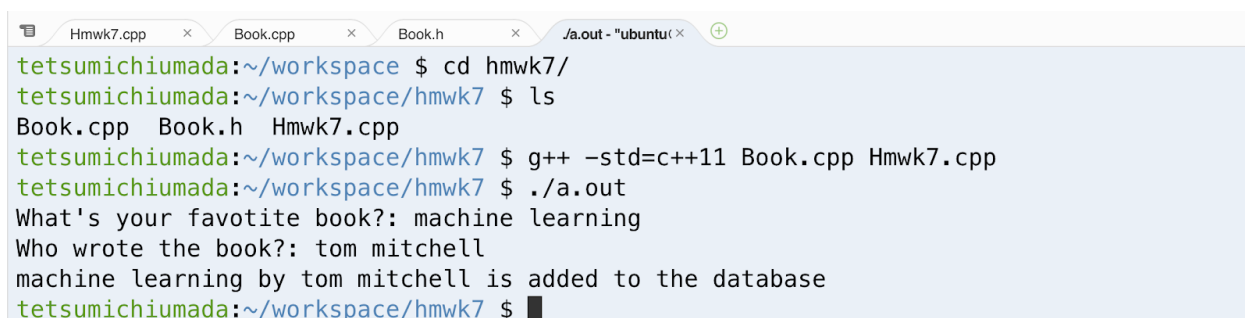
```
g++ -o myExe.out -std=c++11 file1.cpp file2.cpp main.cpp
```

To execute the file, use the following command:

```
./myExe.out
```

### Example1:

Compiling `book.cpp` and `Hmkw7.cpp`. The picture below shows a **bash** window.

A screenshot of a bash terminal window. The terminal has several tabs at the top: 'Hmkw7.cpp', 'Book.cpp', 'Book.h', and 'a.out - ubuntu'. The terminal content shows the following sequence of commands and outputs:

```
tetsumichiumada:~/workspace $ cd hmkw7/
tetsumichiumada:~/workspace/hmkw7 $ ls
Book.cpp  Book.h  Hmkw7.cpp
tetsumichiumada:~/workspace/hmkw7 $ g++ -std=c++11 Book.cpp Hmkw7.cpp
tetsumichiumada:~/workspace/hmkw7 $ ./a.out
What's your favotite book?: machine learning
Who wrote the book?: tom mitchell
machine learning by tom mitchell is added to the database
tetsumichiumada:~/workspace/hmkw7 $
```

Example2: Here, the executable file is named `hwmk7`.

```
Hmwk7.cpp x Book.cpp x Book.h x /hwmk7 - "ubunt x User.cpp x User.h x +
tetsumichiumada:~/workspace $ cd hwmk7/
tetsumichiumada:~/workspace/hwmk7 $ ls
Book.cpp Book.h Hmwk7.cpp User.cpp User.h
tetsumichiumada:~/workspace/hwmk7 $ g++ -std=c++11 Book.cpp User.cpp Hmwk7.cpp -o hwmk7
tetsumichiumada:~/workspace/hwmk7 $ ./hwmk7
Select a numerical option:
=====Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit
6
good bye!
tetsumichiumada:~/workspace/hwmk7 $ █
```

## IMPORTANT: How to submit to autograder

You can find “**Homework 7 - Inginious**” link on Moodle under *Week 9*



## Homework 7 - Inginious

1. Click on the “**Homework 7 - Inginious**” link. Then, bind your INGINious account

CSCI 1300 - Fleming - CS1 Starting Computing

[Home](#) / [My courses](#) / [Fall 2018](#) / [CSCI1300-F18](#) / [Week 9: Classes and Objects](#) / [Homework 7 - Inginious](#)

### Homework 7 - Inginious



Hello!

Welcome on the INGINious platform.

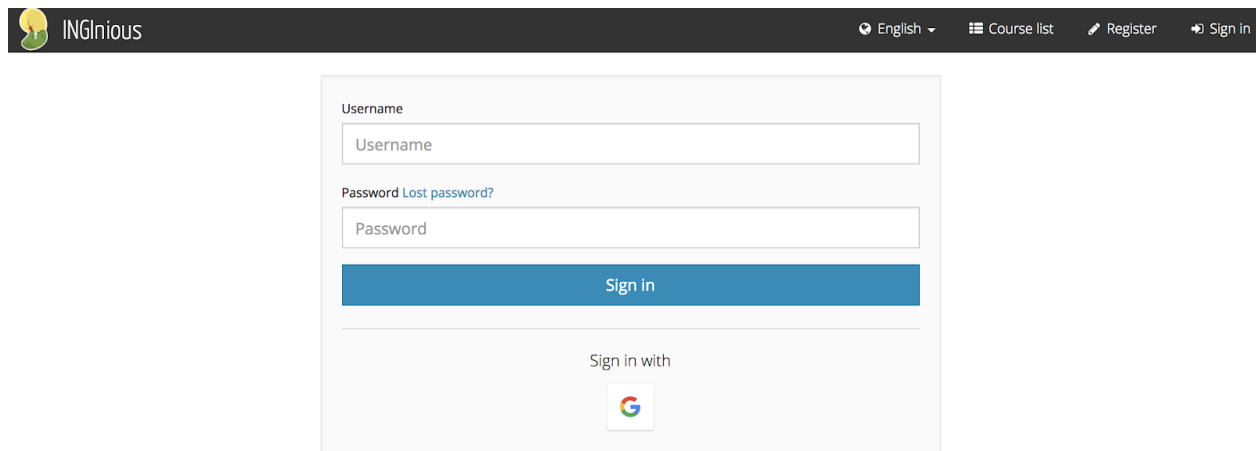
In order to use INGINious via this platform, please bind your existing account or create a new one.

[Bind your INGINious account](#)

[Refresh](#)

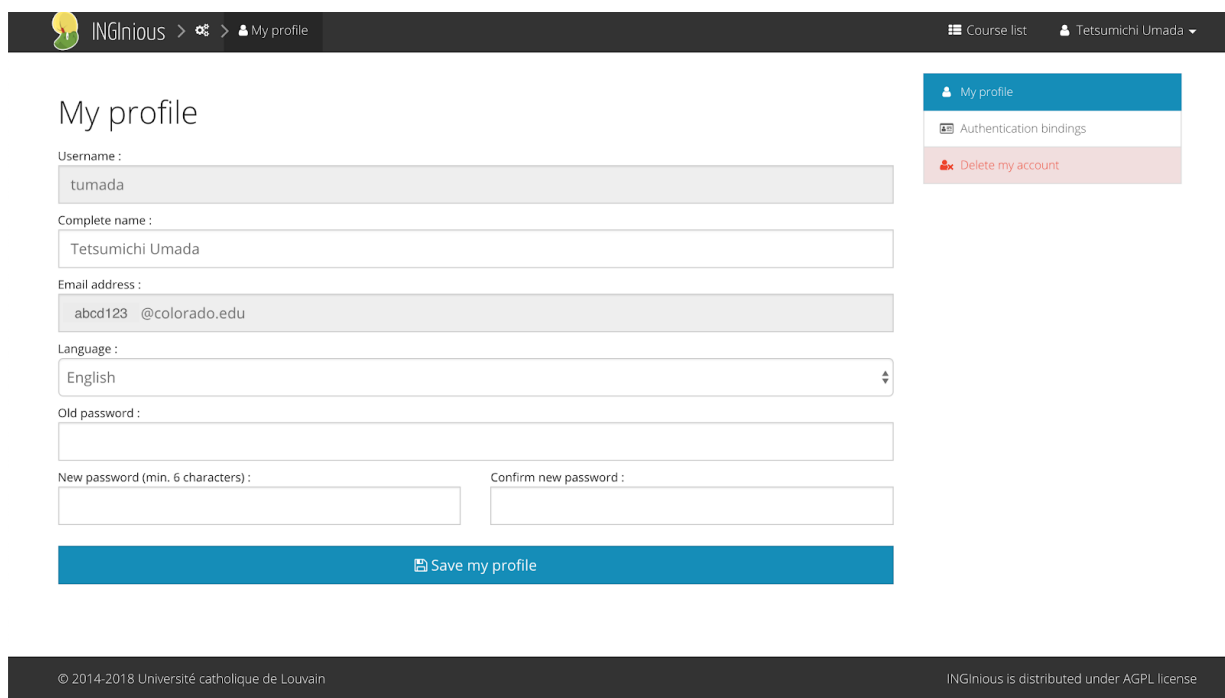
Powered by [INGInious](#), a free and open-source grading tool.

2. Choose “sign in with G”, and log-in with your colorado.edu identity key and password:



The image shows the INGInious login page. At the top, there is a dark header with the INGInious logo on the left and navigation links for 'English', 'Course list', 'Register', and 'Sign in' on the right. The main content area is a light gray box containing a login form. The form has two input fields: 'Username' and 'Password'. Below the password field is a blue 'Sign in' button. Underneath the button is a 'Sign in with' section featuring a Google logo icon.

3. Choose a username:



The image shows the 'My profile' page in INGInious. The header is dark with the INGInious logo, navigation links, and a user profile dropdown showing 'Tetsumichi Umada'. The main content area is titled 'My profile' and contains several input fields: 'Username' (filled with 'tumada'), 'Complete name' (filled with 'Tetsumichi Umada'), 'Email address' (filled with 'abod123 @colorado.edu'), 'Language' (a dropdown menu set to 'English'), 'Old password', 'New password (min. 6 characters)', and 'Confirm new password'. A blue 'Save my profile' button is at the bottom. On the right side, there is a sidebar with three links: 'My profile', 'Authentication bindings', and 'Delete my account'.

4. After making an account, you need to go back to the submission link in Moodle, click it again, then Click Bind my account. You should see a screen similar to the one below. Choose “Bind my account”

## Binding to an existing LMS

Authorization required

You are going to bind your INGINIOUS account 'Your name [username]' with the following LTI tool and context:

- **Tool name:** CS-Moodle
- **Tool description:** Computer Science Moodle
- **Tool URL:** N/A
- **Tool context:** CSC1300-F18 - CSC1 1300 - Fleming - CS1 Starting Computing
- **Tool identifiers:** 6714 - Your name

This operation cannot be undone.

Bind my account

Cancel

### 5. Then, you're ready to submit the zip file

CSCI 1300 - Fleming - CS1 Starting Computing

[Home](#) / [My courses](#) / [Fall 2018](#) / [CSCI1300-F18](#) / [Week 9: Classes and Objects](#) / [Homework 7 - Inginious](#)

Homework 7 - Inginious

Homework7

Homework7

Choose File

No file chosen

Max file size: 1.0MB  
Allowed extensions: .zip

Submit

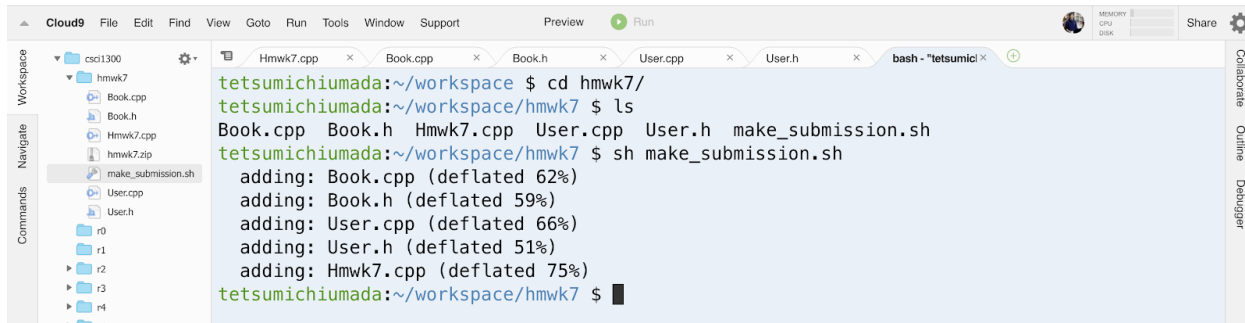
>\_

Powered by INGINIOUS, a free and open-source grading tool.

### 6. What you submit:

- The file name: hmwk7.zip
- The zip file contains: Book.h, Book.cpp, User.h, User.cpp, Hmwk7.cpp
- The name of each file should be the exactly the same (e.g. **B**ook.cpp, **B** is large B).

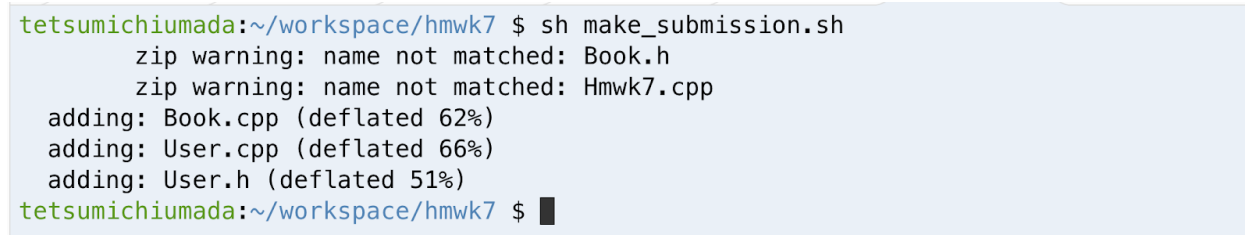
To make sure you won't worry about zipping and submitting the correct files, we created a shell script file you can use: `make_submission.sh`. Simply download the shell script from Moodle, upload it to Cloud9, and run it from your folder to create the `hmwk7.zip` file.



The screenshot shows the Cloud9 IDE interface. On the left, a file explorer shows a workspace with folders `csd1300` and `hmwk7`. The `hmwk7` folder contains `Book.cpp`, `Book.h`, `Hmwk7.cpp`, `User.cpp`, `User.h`, `hmwk7.zip`, and `make_submission.sh`. The main editor shows a terminal window with the following commands and output:

```
tetsumichiumada:~/workspace $ cd hmwk7/
tetsumichiumada:~/workspace/hmwk7 $ ls
Book.cpp Book.h Hmwk7.cpp User.cpp User.h make_submission.sh
tetsumichiumada:~/workspace/hmwk7 $ sh make_submission.sh
adding: Book.cpp (deflated 62%)
adding: Book.h (deflated 59%)
adding: User.cpp (deflated 66%)
adding: User.h (deflated 51%)
adding: Hmwk7.cpp (deflated 75%)
tetsumichiumada:~/workspace/hmwk7 $
```

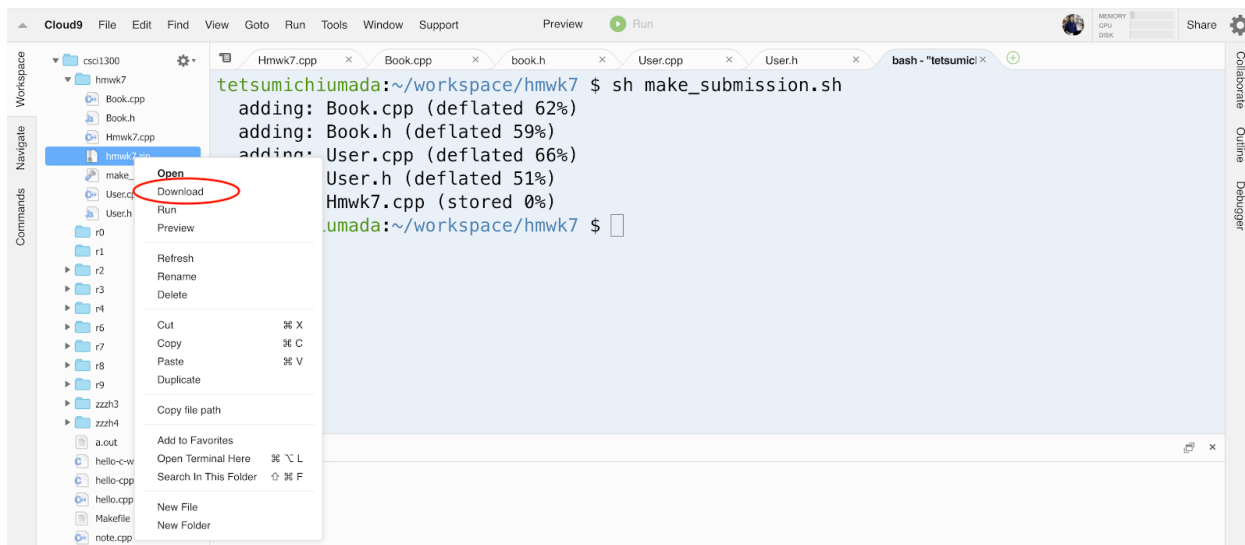
If filename is wrong, it gives a warning message. (it means your zip file does not contain necessary files)



The screenshot shows a terminal window in Cloud9 with the following output:

```
tetsumichiumada:~/workspace/hmwk7 $ sh make_submission.sh
zip warning: name not matched: Book.h
zip warning: name not matched: Hmwk7.cpp
adding: Book.cpp (deflated 62%)
adding: User.cpp (deflated 66%)
adding: User.h (deflated 51%)
tetsumichiumada:~/workspace/hmwk7 $
```

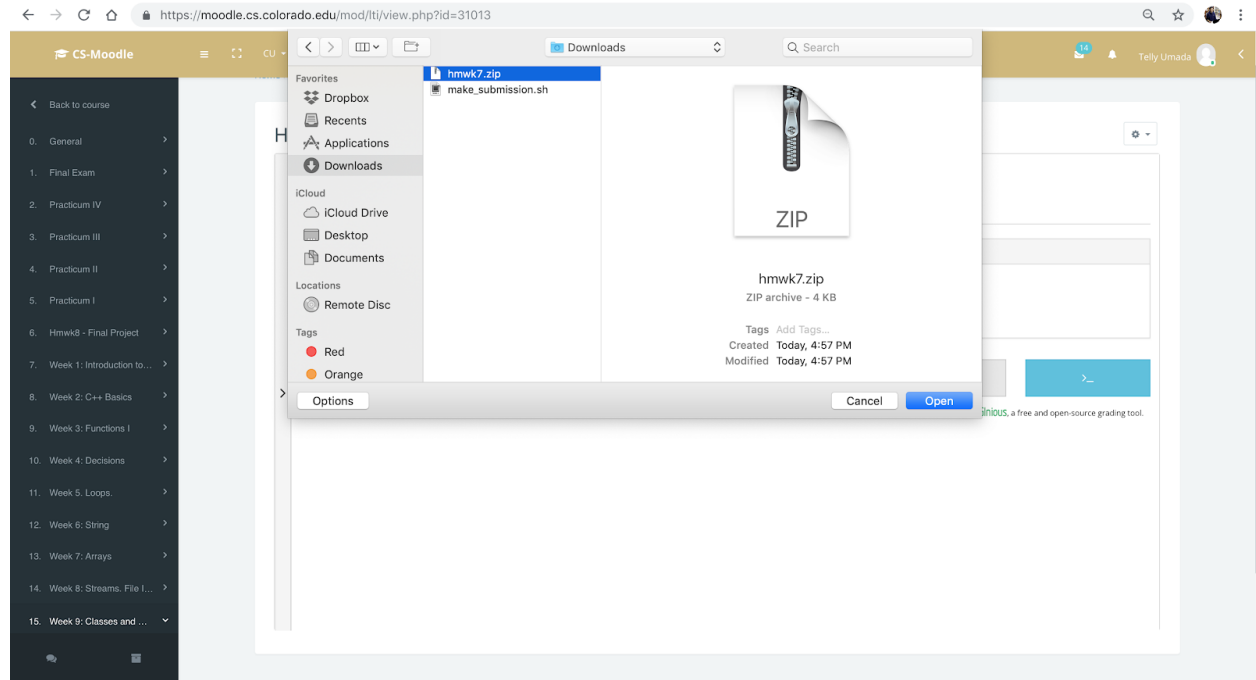
Then, download `hmwk7.zip` and upload it to INGIInious



The screenshot shows the Cloud9 IDE interface. The file explorer on the left shows the `hmwk7` folder. A right-click context menu is open over the `hmwk7.zip` file, with the `Download` option highlighted by a red circle. The terminal window shows the output of the `make_submission.sh` script:

```
tetsumichiumada:~/workspace/hmwk7 $ sh make_submission.sh
adding: Book.cpp (deflated 62%)
adding: Book.h (deflated 59%)
adding: User.cpp (deflated 66%)
adding: User.h (deflated 51%)
Hmwk7.cpp (stored 0%)
umada:~/workspace/hmwk7 $
```

**Note:** Everytime you make a modification, you need to make a new zip file and download a new zip file and upload it to INGIInious.



**Sample output: (user can select 1, 2 or 6. The user input in the bold)**

```
Select a numerical option:
=====Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit
```

**1**

Enter a book file name:

**wrongfile.txt**

No books saved to the database

```
Select a numerical option:
=====Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit
```

**2**

Enter a book file name:

**this\_does\_not\_exist.txt**

No users saved to database

Select a numerical option:

=====Main Menu=====

1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit

**2**

Enter a rating file name:

**ratings.txt**

cynthia...

diane...

joan...

( . . . truncated . . . )

raymond...

adam...

johnny...

Total users in the database: 86

Select a numerical option:

=====Main Menu=====

1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit

**1**

Enter a book file name:

**books.txt**

Total books in the database: 50

Select a numerical option:

=====Main Menu=====

1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit

**3**

```

Here is a list of books
The Hitchhiker's Guide To The Galaxy by Douglas Adams
Watership Down by Richard Adams
The Five People You Meet in Heaven by Mitch Albom
Speak by Laurie Halse Anderson
I Know Why the Caged Bird Sings by Maya Angelou
Thirteen Reasons Why by Jay Asher
Foundation Series by Isaac Asimov
( . . . truncated . . . )
Maus: A Survivor's Tale by Art Spiegelman
The Joy Luck Club by Amy Tan
The Lord of the Rings by J R R Tolkien

```

```

Select a numerical option:
=====Main Menu=====
1. Read book file
2. Read user file
3. Print book list
4. Find number of books user rated
5. Get average rating
6. Quit
6
good bye!

```

### Grading Rubric:

Criteria	Points
Autograder - Ingenious	90
Comments/Style	10
Total	100
Early submission bonus	+ 5% or 2%