# SEG1120 – Data Structures
# Semester 2, 2024

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

**Assignment** *3* **Due Date** *Sunday 3 November 2024 @ 2359*

**Purpose** *To measure the student's ability to solve an underlying problem in C++ by implementing and/or using one or more complex data structures.*

## Introduction

Nestled in the heart of Callaghan, NSW, The Enchanted Quill Bookstore has been a beloved community hub for decades. Founded by the eccentric and passionate bibliophile, Mr. Alistair Quill, the store is renowned for its vast collection of rare and unique books, cozy reading nooks, and enchanting atmosphere. However, as the years passed, the store's inventory system became outdated and inefficient, making it increasingly difficult to manage the ever-growing collection of books.

Mr. Quill, now in his late seventies, realised that the store's charm was being overshadowed by the chaos of its inventory. Books were misplaced, genres were mixed up, and customers often left frustrated. Determined to restore order and enhance the customer experience, Mr. Quill sought help from the brightest minds in the community.

Enter the students of SENG1120, a group of talented and enthusiastic students from the University of Newcastle. Recognising their potential and innovative spirit, Mr. Quill approached their course coordinator with a unique proposition: to develop a more modern inventory management system for The Enchanted Quill.

The students were tasked with creating an inventory management system that would utilise a binary search tree to store detailed information about each book, ensuring quick and efficient access. Additionally, they would implement a hash table with separate chaining to categorise books by genre, allowing for seamless retrieval.

By leveraging their skills in data structures and algorithms, the students aimed to transform The Enchanted Quill into a model of efficiency and order. Their system would enable Mr. Quill and his staff to easily add new books, remove outdated ones, and query information with ease, ultimately preserving the magic of the bookstore for generations to come.

And so, the students of SENG1120 embarked on their journey, eager to apply their knowledge and make a meaningful impact on their community.

## Assignment Overview

This assignment will test your ability to implement a hash table and binary search tree, then use their functionality to implement an inventory management system that will maintain information about books. The inventory management program will use a binary search tree to store book details and a hash table (with separate chaining) to store information about the different book genres. The inventory management system will then have basic functionality to add books, remove books, and query various information.

This assignment is worth 100 marks and accounts for 15% of your final grade. Late submissions are subject to the rules specified in the Course Outline.

## The Supplied Files

This section gives an overview of the files that you are provided as part of this assignment, none of which you should modify. You are not provided with (skeleton) implementation files – you will be expected to create these files from scratch. You are recommended to take some time to understand the overall structure of the program before starting your implementation.

- **`main.cpp`** – contains the main function and the logic to initialise the inventory management system and perform basic operations. **This file should not be modified.**
- **`book.h`** – contains the header for the book class, which includes the instance variables and behaviours that a book contains. **This file should not be modified.**
- **`book_genre_stats.h`** – contains the header for the `book_genre_stats` class, which includes the instance variables and behaviours for the object that maintains information about a particular genre of book. **This file should not be modified.**
- **`binary_search_tree.h`** – contains the header for the `binary_search_tree` class, which includes the instance variables and behaviours that the binary search tree contains. **This file should not be modified.**
- **`hash_table.h`** – contains the header for the `hash_table` class, which includes the instance variables and behaviours that the hash table contains. **This file should not be modified.**
- **`inventory.h`** – contains the header for the `inventory` class, which includes the instance variables and behaviours that the inventory management system contains. **This file should not be modified.**
- **`makefile`** – the `makefile` for the project, which outlines the build recipe. The `-g` flag is set for you to make debugging and/or memory leak detection more convenient, should you wish to use these tools. **This file should not be modified.**

- `books.csv` – a CSV file containing a list of 250 books[1]. Each book has an ISBN, a title, a genre, and a rating, and power level. **This file should not be modified. You may consider using a smaller version for testing if you wish.**

## Running the Program

Of course, you will need to ensure the program is compiled prior to running it. You can use the supplied `makefile` for this – it will produce an executable called `Assignment3`. The program can be executed as normal using the command:

```
./Assignment3
```

This will read a file named `books.csv` to populate the inventory with the data for 250 books. **This file must be present for the code to run.** Several operations are then performed to test its functionality. However, this testing is not exhaustive, and you are strongly encouraged to test the broader functionality of your program and, in particular, your tree and hash table. An example of the program execution is shown in Figure 1, Figure 2, and Figure 3 – it is provided in 3 separate figures given the length of output.

**Note:** Not all output is captured in these screenshots.

**Note:** The program will not compile as supplied, as you will not have the necessary implementation files. You are encouraged to write skeleton files, like those provided in Assignment 1, to allow compilation of an incomplete program – this will allow you to work incrementally.

## Part 1: Implementation

The first (and simplest) task is to implement the `book` and `book_genre_stats` classes. These classes will require a few operators to be overloaded such that they can be used in the tree and hash table, as well as be inserted into a stream for printing.

The next task is to implement the templated `binary_search_tree`, and `hash_table` classes. For each class, you are provided with the header file and must adhere to its definitions. These headers will be substantially similar to those discussed in lecture and lab but may differ in a few minor ways. In particular, the hash table will use `std::list` for separate chaining and a few function definitions have been modified and/or added for completeness.

Finally, you will use your `binary_search_tree` and `hash_table` classes to provide the expected functionality for the `inventory` class.

[1] This assignment uses a modified version of the book data from here: Best Books Ever Dataset (zenodo.org)

Some additional details about each class are provided below, but you should also examine the documentation in the provided files, which also contains important information about the expected behaviour.

## book

The book class is a simple data class that stores information about a book, particularly its ISBN, genre, title, and a rating. For a full list of methods required and some important details, you should examine the **`book.h`** file and its associated documentation.

## book_genre_stats

The book_genre_stats class is a simple data class that stores information about a specific genre of book, particularly the genre, a count of the number of books of that genre, and the total cumulative rating of all books of that genre. This should align with the current inventory. That is, if a book is added or removed from the inventory tree, it's corresponding stats object should be updated accordingly. For a full list of methods required and some important details, you should examine the **`book_genre_stats.h`** file and its associated documentation.

## binary_search_tree

The binary_search_tree class is a templated version of a binary search tree and should be implemented recursively as discussed in lecture. For a full list of methods required and some important details, you should examine the **`bs_tree.h`** file and its associated documentation.

By default, your binary search tree should print an inorder traversal but will provide functionality to print according to all three traversals discussed in lecture (namely preorder, postorder, and inorder).

## hash_table

The hash_table class is a templated version of a hash table that uses a linked list for separate chaining and should be implemented as discussed in lecture. For a full list of methods required and some important details, you should examine the **`hash_table.h`** file and its associated documentation.

**Note:** you will notice that **`hash_table.h`** contains a few helper methods implemented for you. For example, the hash function is already implemented for you. Additional helper methods are provided to support operations you will need to perform with the `std::list`, such as finding, removing, and printing items. You are encouraged to review these methods, and use them where appropriate. Further details about when a helper method may be appropriate are given in **`hash_table.h`**.

### inventory

The inventory class contains the main logic of the inventory management system and uses your binary_search_tree and hash_table classes to support the functionality, as further detailed in the header file. Your hash table should be initialised with 101 cells, as is the default size in the header file, but should work for any provided capacity. For a full list of methods required and some important details, you should examine the **inventory.h** file and its associated documentation.

The add_book method will accept a new book object and should then update the inventory accordingly. This includes both inserting the book object into the tree and adding or updating a book_genre_stats object in the hash table. Conversely, the remove_book method should remove the book object from the tree and update the book_genre_stats object accordingly. Particularly, the book_genre_stats object should be removed from the hash table when no books of its genre remain.

There are a few remaining methods that will be provided to support information querying from the inventory.

## Part 2: Report

In addition to a working implementation, you will provide a written report (worth 15% of the assessment grade) that discusses and evidences your implementation and testing processes. The report should outline your use of the Problem Analysis-Coding-Execution cycle. The report is expected to be 1-2 pages in length (maximum 12-point font), but there isn't really a strict page limit. However, the report is not meant to be a major endeavour, but rather is to prompt your thinking and ensure that you have considered your solution in a bit more depth than just getting the code to compile and submitting your first draft.

Your report should include the following sections:

1. **Implementation:** While the design has been provided for you, you will discuss how you arrived at your implementation of the various classes. In particular, include a high-level description of how the code works. You should include references to the lecture material (i.e., slide numbers), where appropriate, as evidence of how you arrived at a particular implementation.

   We don't need a detailed discussion of every method, but rather a 1-2 paragraph discussion of the implementation of the entire system, largely to convince us that you understand *what* you did and *why* you did it.

2. **Testing:** Describe how you tested your implementation. Describe any test cases you conducted and discuss the results. The main point is to prompt you to consider some simple tests that cover the functionality, ensuring that the code works as expected.

You don't necessarily need to test and report on every method but should ensure that you have convinced yourself (and the marker) that you have a working solution beyond the functionality shown in the provided **`main.cpp`**.

This may be easiest to provide in tabular form, such as below:

| Test ID | Test Description | Expected Result | Test Result |
|---|---|---|---|
| 1 | … | … | … |

3. **Reflection:** Reflect on what you learned from this assignment and how you could improve your implementation. For example, are there areas where error checking would make sense, but were not included in the specification? Is there a different design you think would be better? Have you learned about a data structure that would be more efficient for some of the operations?

   Again, we are mainly just looking for a 1-2 paragraph summary of how your skills were improved through this assignment and if you have any ideas for ways things could be improved in your solution.

   You can find more information about reflective writing in the library guide found here: What is Reflective Writing? - Reflective writing and blogs - LibGuides at University of Newcastle Library.

## Marking

Your implementation will account for 85 marks and will be assessed on both correctness and quality. This means, in addition to providing a correct solution, you are expected to provide readable code with appropriate commenting, formatting, best practices, memory management, etc. **Code that fails to compile will likely result in a zero for the functionality correctness section.** At the discretion of the marker, minor errors may be corrected (and penalised) but there is no obligation, nor should there be any expectation, that this will occur. Your code will be tested on functionality not explicitly shown in the supplied demo file (i.e., using a different **`main.cpp`** file). Hence, you are encouraged to test broader functionality of your program before submission, particularly as you will need to discuss this in your written report.

Your report will account for 15 marks (5 marks for each of the required sections) and will be assessed on general quality and maturity of the provided information and discussions.

Marking criteria will accompany this assignment specification and will provide an indicative guideline on how you will be evaluated. **Note**: the marking criteria is subject to change, as necessary.

## Submission

Your submission should be made using the Assignment 3 link in the Assignments section of the course Canvas site. Assignment submissions will not be accepted via email. Incorrectly submitted assignments may be penalised.

Your submission should include only the completed versions of **`book.cpp`**, **`book_genre_stats.cpp`**, **`inventory.cpp`**, **`binary_search_tree.hpp`**, and **`hash_table.hpp`**. You do not need to include any other files in your submission. Be sure that your code works with the supplied files. Do not change the files we have supplied as your submission will be expected to work with these files – when marking your code, we will add the required files to your code and compile it using the supplied **`makefile`**.

Compress the required files into a single **`.zip`** file, using your student number as the archive name. Do not use **`.rar`**, **`.7z`**, **`.gz`**, or any other compressed format – these will be rejected by Canvas. For example, if your student number is c9876543, you would name your submission:

**`c9876543.zip`**

If you submit multiple versions, Canvas will likely append your submission name with a version number (e.g., **`c9876543-1.zip`**) – this is not a concern, and you will not lose marks.

Remember that your code will conform to C++ best practices, as discussed in lecture, and should compile and run correctly using the supplied **`makefile`** in the standard environment (i.e., the Debian virtual machine). **If you have developed your solution using any other environment, be sure to test it with the VM prior to submission.** There should be no segmentation faults or memory leaks during or after the execution of the program. Please see the accompanying marking criteria for an indicative (but not final) guideline to how you will be evaluated.

## Helpful Tips

A few tips that may help you along your journey.

1. Read the header files carefully – they provide further information on the specification for various methods.

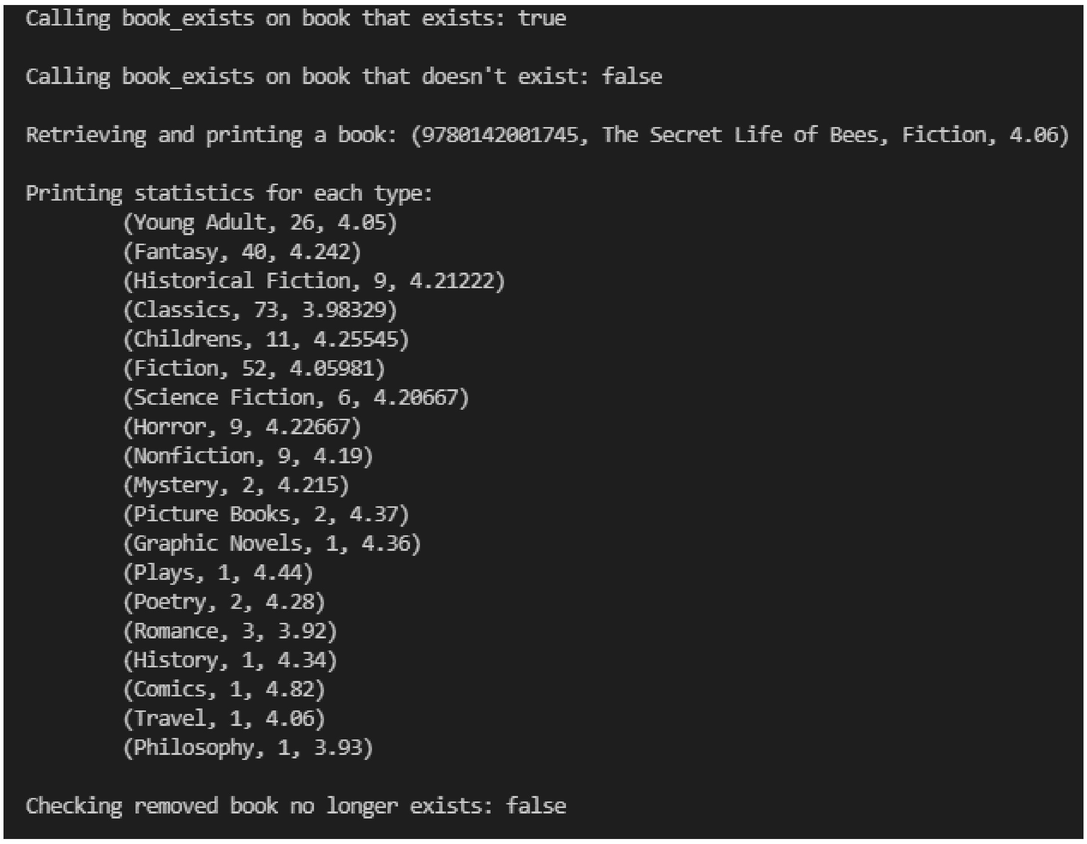
2. Work incrementally – don't try to implement everything at once. Consider getting a barebones version to compile, which will enable you to test methods as you implement them.

3. Remember that you can debug your program in VS Code. See Lab3a for a brief guide. Of course, the name of the executable in your **`launch.json`** file will be different than the example in the lab, but this gives you a great way to inspect your data structures during the program execution. You will need to think carefully about what you would expect the data to look like.

4. You will be expected to supply a program with no memory leaks. You are encouraged to use `valgrind` to assess whether you have any leaking memory in your program.

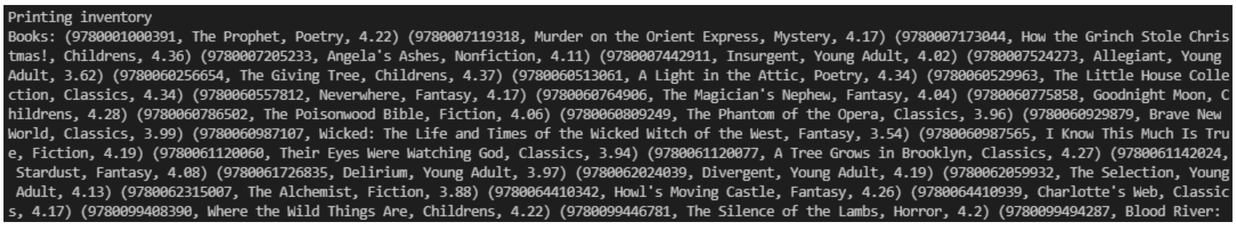
5. Start early! The longer you wait to start, the less time you will have to complete the assignment. This is particularly important for Assignment 3, as the due date is immediately before the examination period – hence, it is in your best interest to complete the assignment ASAP to allow sufficient time to prepare for your examinations.

## Good Luck!



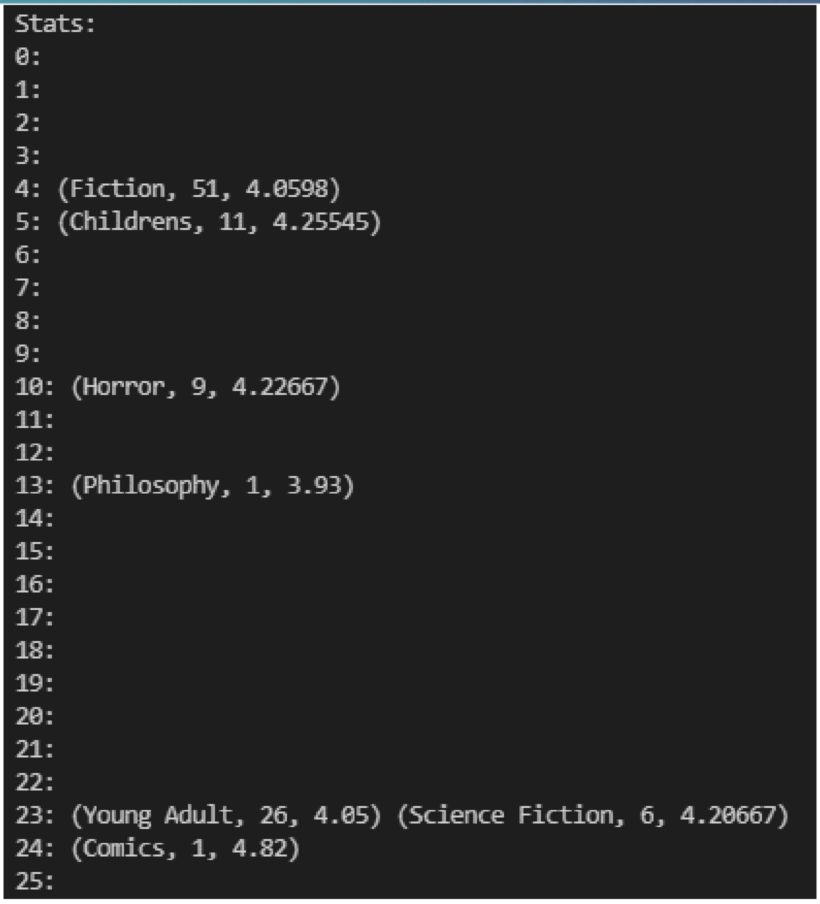
```
Calling book_exists on book that exists: true

Calling book_exists on book that doesn't exist: false

Retrieving and printing a book: (9780142001745, The Secret Life of Bees, Fiction, 4.06)

Printing statistics for each type:
        (Young Adult, 26, 4.05)
        (Fantasy, 40, 4.242)
        (Historical Fiction, 9, 4.21222)
        (Classics, 73, 3.98329)
        (Childrens, 11, 4.25545)
        (Fiction, 52, 4.05981)
        (Science Fiction, 6, 4.20667)
        (Horror, 9, 4.22667)
        (Nonfiction, 9, 4.19)
        (Mystery, 2, 4.215)
        (Picture Books, 2, 4.37)
        (Graphic Novels, 1, 4.36)
        (Plays, 1, 4.44)
        (Poetry, 2, 4.28)
        (Romance, 3, 3.92)
        (History, 1, 4.34)
        (Comics, 1, 4.82)
        (Travel, 1, 4.06)
        (Philosophy, 1, 3.93)

Checking removed book no longer exists: false
```

*Figure 1. Example execution of the program, only showing the first few lines of output.*



```
Printing inventory
Books: (9780001000391, The Prophet, Poetry, 4.22) (9780007119318, Murder on the Orient Express, Mystery, 4.17) (9780007173044, How the Grinch Stole Chris
tmas!, Childrens, 4.36) (9780007205233, Angela's Ashes, Nonfiction, 4.11) (9780007442911, Insurgent, Young Adult, 4.02) (9780007524273, Allegiant, Young
Adult, 3.62) (9780060256654, The Giving Tree, Childrens, 4.37) (9780060513061, A Light in the Attic, Poetry, 4.34) (9780060529963, The Little House Colle
ction, Classics, 4.34) (9780060557812, Neverwhere, Fantasy, 4.17) (9780060764906, The Magician's Nephew, Fantasy, 4.04) (9780060775858, Goodnight Moon, C
hildrens, 4.28) (9780060786502, The Poisonwood Bible, Fiction, 4.06) (9780060809249, The Phantom of the Opera, Classics, 3.96) (9780060929879, Brave New
World, Classics, 3.99) (9780060987107, Wicked: The Life and Times of the Wicked Witch of the West, Fantasy, 3.54) (9780060987565, I Know This Much Is Tru
e, Fiction, 4.19) (9780061120060, Their Eyes Were Watching God, Classics, 3.94) (9780061120077, A Tree Grows in Brooklyn, Classics, 4.27) (9780061142024,
 Stardust, Fantasy, 4.08) (9780061726835, Delirium, Young Adult, 3.97) (9780062024039, Divergent, Young Adult, 4.19) (9780062059932, The Selection, Young
 Adult, 4.13) (9780062315007, The Alchemist, Fiction, 3.88) (9780064410342, Howl's Moving Castle, Fantasy, 4.26) (9780064410939, Charlotte's Web, Classic
s, 4.17) (9780099408390, Where the Wild Things Are, Childrens, 4.22) (9780099446781, The Silence of the Lambs, Horror, 4.2) (9780099494287, Blood River:
```

*Figure 2. Example execution of the program, showing the tree printed. Only some output is shown, given its length. Note that, for an inorder traversal (as is default), the books will be alphabetical order by their ISBN! Each entry indicates the book's ISBN, title, genre, and rating.*



```
Stats:
0:
1:
2:
3:
4: (Fiction, 51, 4.0598)
5: (Childrens, 11, 4.25545)
6:
7:
8:
9:
10: (Horror, 9, 4.22667)
11:
12:
13: (Philosophy, 1, 3.93)
14:
15:
16:
17:
18:
19:
20:
21:
22:
23: (Young Adult, 26, 4.05) (Science Fiction, 6, 4.20667)
24: (Comics, 1, 4.82)
25:
```

*Figure 3. Example execution of the program, showing the stats hash table being printed. Lines with multiple entries indicate collisions, which are stored in a list. Only some output is shown, given its length. Each entry indicates information about the genre. For example, the entry at index 4 indicates there are 51 books in the Fiction genre. Together, these 51 books have an average rating of 4.0598.*