Ryan Craddock

**Final Report**

The project I worked on is called Library Management System (LMS). It was designed to manage the book inventory of a library. Its goal was to implement features to check out and return books, make reservations, keep track of overdue books, and charge fines. The system has been implemented in C++ to embrace object-oriented programming concepts, file I/O, and suitable memory management. We wanted to design a tool that has utility for librarians as they maintain their collections and reach out to library patrons.

The system was built around a few classes, each with its functions and responsibilities. The `Book` class surveys books from the inventory. ISBN, title, author, genre, year of publication, and availability status are among its key features. The class was expected to have default and parameterized constructors which initialize these attributes. The default constructor initializes the availability status to true, indicating that books are available for checkout when first created, while the parameterized constructor enables more specific initialization. I added other getter and setter methods to get and set the attributes of the book while keeping the concept of encapsulation. A really important part of the `Book` class was the `getFormattedDetails` function, which returns a string containing the formatted display of that book's details and simplifies presenting this information to the librarian.

The following significant part was the `Inventory` class which took care of the set of books. Data members: For the array of Book objects, I used an array of pointers to Book objects to handle dynamic memory allocation and offer flexibility in managing the books added to the store. The `Inventory` class itself needed to support multiple operations, such as adding and removing books, querying books by their ISBN, and updating the availability of books. I also used a hashing technique in the `LibraryHash` class to make operations like insert and look up more efficient by mapping each book to an index in the array using its ISBN. It allows books to be stored as efficiently as possible and provides an O(logn) time complexity for retrieval and removal operations. To facilitate the librarian in managing the inventory and interacting with the system functions like addBook, removeBook, and listAvailableBooks were created. Methods such as, `countTotalBooks`, and `updateBookAvailability` were written to count the number of books in the system and ensure that the books were accurately checked in or out.

The `LibraryHash` class is a pivotal part of the system, providing methods for hashing ISBNs and formatting ISBN strings. `HashBook` Function I wrote the `HashBook` function to generate a hash value based on a book’s ISBN, which is then used to determine where the book will be stored in the inventory. Since I was manually generating the hashes, I also had to make sure that the ISBNs were correctly formatted for the hash function which is where the `format ISBN` method came into play as it stripped out any non-numeric characters in the ISBN string and stores the entire ISBN string as a long integer. It allows us to ensure that the ISBN is always in the right format and avoid potential issues when we send it to the hashing function.

Since I developed this process I faced a few difficulties. By far one of the biggest challenges, though, was making sure that the hash function spread all the books evenly throughout the inventory. Collision management was quite difficult at first, but once I reconfigured the hash function to use a modulo-divided approach, I reduced some of these kinks and got a decent uniform distribution of books. Managing memory was another challenge, particularly since I was working with dynamic memory allocation. Particularly, I needed to make sure that no memory was leaking. This was important to keep in mind, particularly in the destructor for the `Inventory` class which I have attached here, which releases the dynamic memory that was allocated when the object is destroyed. Another hurdle was incorporating file I/O for reading and writing the inventory data. Not only that, but I also had to create a system that could accurately parse the CSV file with all the book details and store it in the inventory. I had to consider errors in the file data, such as missing or malformed information, and write exception handling around that code such that the system does not crash in these situations.

Despite these issues, the project provided some important lessons. I built upon my knowledge of object-oriented programming principles, such as encapsulation, inheritance, and polymorphism, which were critical in designing a modular and maintainable system. Further, I became more adept with memory management in C++ (like dynamic memory allocation and de-allocation). I learned also how to design classes with a single responsibility and how to make sure that they cooperate as expected. Since the system is designed modularly, the interaction between the `Book`, `Inventory`, and `LibraryHash` classes was seamless, so by keeping the class boundaries clear, I was able to stay away from unnecessary complexity.

In conclusion, the overall development of the Library Management System was a mindfully satisfying learning process. Through this project, I was able to leverage the core principles of programming to build a working system that can handle a library’s stock and help librarians in their day-to-day activities. Although I learned a lot during the process (for instance, hashing, integrating memory management, and file I / O), I encountered issues along the way. I would love to keep improving the system, adding a user interface and advanced file-handling features to make it more comprehensive in the future.