Semana 2: BookStore (C++)

1) Purpose & Learning Goals

Build a console-based **BookStore** app that supports:

- Operator overloading (Book::operator<)
- Manual **binary search** (title and ISBN)
- STL containers: std::array, std::vector, std::list
- Input parsing & simple auditing/logging

2) Description

The program manages an in-memory catalog of books and allows the user to:

1. Insert books
2. Search by title (binary search)
3. Search by ISBN (binary search)
4. Show the audit log

Books are stored in a std::vector<Book> named catalogue. The **catalog must remain sorted by title** at all times to guarantee correctness of the title binary search.

The audit is a std::list<std::string> (audit) where each operation pushes a human-readable message (most recent first).

3) Data Model

3.1 Book

```cpp
class Book {
public:
    std::string title;
    std::string author;
    std::array<int, 13> isbn{};   // 13 digits: 0–9

    Book();
    Book(std::string title, std::string author, std::array<int,13> isbn);

    // Natural ordering: (title, author, isbn)
    bool operator<(const Book& other) const;
};
```

Operator<

Rationale: Title-first ordering supports the title binary search. Author/ISBN are tie-breakers.

4) System Architecture

4.1 BookStore

Core responsibilities:

- Maintain catalogue sorted by title
- Insert books
- Search books (by title, by ISBN)
- Keep an audit log
- Utility helpers (timestamp, string→ISBN conversion, etc.)

```cpp
class BookStore {
public:
    std::vector<Book> catalogue;  // must be sorted by title (invariant)
    std::list<std::string> audit;

    BookStore();                          // seeds initial data & sorts
    void insertBook(Book b);              // keeps catalogue sorted
    int  searchBookByTitle(std::string title);// binary search over catalogue
    int  searchBookByISBN(std::string isbn);  // binary search via ISBN order
    std::vector<int> sortIndicesByIsbn(const std::vector<Book>&);
    std::array<int,13> stringToIsbnArray(const std::string& input);
    std::string getCurrentTimestamp();
};
```

4.2 Additional Info

- **Catalogue sort (by title):**
  After any insertion, catalogue is re-sorted with std::sort (uses Book::operator<).
- **Binary search preconditions:**
  - searchBookByTitle: catalogue is sorted **by title**.
  - searchBookByISBN: do **not** assume catalogue is sorted by ISBN; build an **ISBN-ordered collection** and binary-search **that collection**.
- **ISBN validity:** stringToIsbnArray must yield exactly 13 digits or throw/indicate error.

5) Required Functions (Format & Behavior)

5.1 insertBook

- Appends the book and re-sorts the catalogue by title (using std::sort).

- Adds an audit line: "Book added: <title>"

5.2 searchBookByTitle

- Implements manual binary search over catalogue (sorted by title).
- On found: push an audit line
  "Search by title: <title> FOUND. <timestamp>"
  and return the index.
- On not found: push
  "Search by title: <title> NOT FOUND. <timestamp>"
  and return -1.

5.3 sortIndicesByIsbn

std::vector<int> BookStore::sortIndicesByIsbn(const std::vector<Book>& books)

- Produces a vector of indices [0..n-1], sorted by books[i].isbn (lexicographic).
- Used to build an ISBN-ordered view without reordering catalogue.

5.4 searchBookByISBN

- Converts input with ***stringToIsbnArray***.
- Build an ISBN-sorted index vector (**sortedIndices**) using **sortIndicesByIsbn**(catalogue)
  .
- Binary-search over **sortedIndices**, comparing ***catalogue[sortedIndices[mid]].isbn*** to the key.
- On found: return the **real index** in catalogue (*sortedIndices[mid]*) and log an audit line "Search by ISBN: <isbn> FOUND. <timestamp>".
- On not found: log "Search by ISBN: <isbn> NOT FOUND. <timestamp>" and return -1.