Open Ended Lab -1

Spring 2025

Course Title: Data Structures Lab

Course Code: CSE1302

Section: 4

Submitted by:

Student Name: Md Tazminur Rahman Tanim

ID: 242014124

Department of CSE

University of Liberal Arts Bangladesh (ULAB)

Library Management System

This lab report provides an overview of a simple Library Management System that has been created using the C programming language. This system utilizes basic data structures like arrays to store book information and essential algorithms like insertion sort for sorting the records and binary search for efficient searching.

The system is also improved to make use of files in that data could be saved and reused even after a program is terminated. This enables the application to persist book records across sessions, rendering the system viable and dynamic.

Key Features Implemented:

- Adding new books to the collection
- Removing existing books from the system
- Displaying all books in ascending order based on Book ID
- Searching for a book by its ID using binary search
- A menu-driven interface for continuous user interaction until exit

Use of Structures:

The program uses a struct named **library** to group related attributes of each book. These attributes include:

- **id**: A unique integer identifier for each book.
- **title**: A character array used to store the title of the book.
- **author**: A character array used to store the name of the author.

This structure provides a clear and organized way to represent a single book's data. By using an array of this structure, the program can perform operations on multiple book records efficiently. It allows for easy addition, removal, searching, and sorting of book entries. This method not only improves code readability but also helps in managing the book collection in a logical and structured manner.

Array of Structures

An array of structures named books (with a fixed size of 100) is used to store multiple book records. This array allows the user to store and manage multiple book details using a single variable. It simplifies the code and improves readability, making the system more user-friendly.

- **Purpose**: Serves as the primary storage for all records in the library system.
- **Operations**: All core actions—adding, deleting, and searching for books—are performed within this array.
- **Index Management**: A global variable `n` is used to keep track of the current number of books stored in the system.

Array Operations in the System

The program is divided into several functions, each responsible for a specific operation. These include:

- **main()**: Manages the overall flow of the program and handles user inputs. It displays a menu-driven interface, prompts the user to make a selection, and then calls the appropriate function based on the user's choice. The loop continues until the user chooses to exit the program.

```c
int main()
{
    int choice, id;
    loadfile();
    printf("Library Management System\n\n");

    while (1)
    {
        printf("**********************************\n");
        printf("1. Add Book\n");
        printf("2. Remove Book\n");
        printf("3. Search Book by ID\n");
        printf("4. Display All Books\n");
        printf("5. Exit\n");
        printf("**********************************\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        if (choice == 1)
        {
            add_books();
        }
        else if (choice == 2)
        {
            remove_books();
        }
        else if (choice == 3)
        {
            printf("Enter book ID to search: ");
            scanf("%d", &id);
            binary_search(id);
        }
        else if (choice == 4)
        {
            display_books_sorted();
        }
        else if (choice == 5)
        {
            printf("Exiting program...\n");
            return 0;
        }
        else
        {
            printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

```
Data loaded from hameem.txt successfully.
Library Management System

**********************************
1. Add Book
2. Remove Book
3. Search Book by ID
4. Display All Books
5. Exit
**********************************
Enter your choice:
```

savefile() and loadfile():

These two functions handle **data persistence** by saving and loading book records to and from a file. The program stores all data in a Notepad file named **"Hameem.txt"**, allowing the system to permanently retain the book records even after the program is closed.

- **savefile()**: Writes the current list of books from the array into the file. This ensures all new entries, deletions, and changes are saved for future use.
- **loadfile()**: Reads previously saved data from the file and loads it into the program when it starts.

```
void savefile()
{
    FILE *file= fopen("hameem.txt","w");
    if(file == NULL)
    {
        printf ("Error opening file\n");
        return;
    }
    for(int i=0; i<n; i++)
    {
        fprintf(file,"%d\t\t%s\t\t\t%s\n", books[i].id,books[i].title,books[i].author);
    }
    fclose(file);
    printf("Data saved to hameem.txt successfully.\n");
}
void loadfile()
{
    FILE *file= fopen("hameem.txt","r");
    if(file == NULL)
    {
        printf ("No existing data file found\n");
        return;
    }
    n=0;
    while(fscanf(file, "%d\t%[^\t]\t\t%[^\n]\n", &books[n].id, books[n].title, books[n].author) == 3)
    {
        n++;
    }
    fclose(file);
    printf("Data loaded from hameem.txt successfully.\n");
}
```

• **add_books():** Adds a new book in the library record. Asks the user the user to input the book id, book name and author name stores it in the function savefile() . The input is taken using a menu driven user interface.

```
void add_books()
{
    printf("Enter an Id for book:");
    scanf("%d",&books[n].id);
    printf("Enter the book title:");
    getchar();
    scanf("%[^\n]%*c", books[n].title);
    printf("Enter the author name:");
    getchar();
    scanf("%[^\n]%*c", books[n].author);
    n++;
    printf("Book added successfully\n");
    savefile();
}
```

```
Enter an Id for book: 101
Enter the book title:The Great Gatsby
Enter the author name:Gabriel García Márquez
Book added successfully
Data saved to hameem.txt successfully.
```

• **remove _books():** Deletes a book record. The user is prompted for the book ID. The system searches the array for the matching ID. Once found, the array elements are shifted left to remove the gap. Then n is decremented and the file is updated.

```
void remove_books()
{
    int id,found=0;
    printf("Enter the book ID to delete :");
    scanf("%d",&id);
    for(int i=0; i<n; i++)
    {
        if(books[i].id==id)
        {
            found=1;
            for(int j=i; j<n-1; j++)
            {
                books[j]=books[j+1];
            }
            n--;
            printf("%d book deleted successfully\n",id);
            savefile();
            return;
        }
    }
    if(found==0)
    {
        printf("Book not found");
    }
}
```

```
Enter the book ID to delete :101
101 book deleted successfully
Data saved to hameem.txt successfully.
```

binary_search(int id):

This function uses the **binary search algorithm** to quickly find a book by its **ID**. Since binary search requires sorted data, the system first applies **insertion sort** to the book list.

The algorithm:

- Splits the array in half,
- Compares the middle element with the target ID,
- Repeats the process in the appropriate half.

With a time complexity of **O(log n)**, it is highly efficient for large datasets.

```c
void binary_search(int id)
{
    sort();
    int first=0, last=n-1,mid;
    while(first<=last)
    {
        mid=(first+last)/2;
        if(books[mid].id<id)
            first=mid+1;
        else if(books[mid].id==id)
        {
            printf("Book found: ID = %d, Title = %s, Author = %s\n",
                   books[mid].id, books[mid].title, books[mid].author);
            return;
        }
        else
            last=mid-1;
        mid=(first+last)/2;
    }
    printf("Book not found in the list\n");
}
```

```c
void sort()
{
    for(int i = 1; i < n; i++)
    {
        struct library t = books[i];
        int j = i - 1;
        while(j >= 0 && books[j].id > t.id)
        {
            books[j + 1] = books[j];
            j--;
        }
        books[j + 1] = t;
    }
}
```

```
Enter book ID to search: 105
Book found: ID = 105, Title = The Great Gatsby, Author = abriel García Márquez
```

• **display_books_sorted():** This function displays all the books in a sorted order. The books are sorted in ascending. And then is display the list along with the book id, title and author name.

```c
void display_books_sorted()
{
    if (n == 0)
    {
        printf("No books available.\n");
        return;
    }

    sort();

    printf("\nList of Books:\n");
    printf("-----------------------------------------------------\n");
    printf("ID\t\tTitle\t\t\tAuthor\n");
    printf("-----------------------------------------------------\n");

    for (int i = 0; i < n; i++)
    {
        printf("%d\t\t%s\t\t%s\n", books[i].id, books[i].title, books[i].author);
    }

    printf("-----------------------------------------------------\n");
}
```

```
List of Books:
-----------------------------------------------------
ID              Title                   Author
-----------------------------------------------------
101             Silent River            Marcus Lee
102             Broken Light            Nina Hart
103             Fading Echo             Liam Grant
-----------------------------------------------------
```

Conclusion:

The Library Management System effectively demonstrates the use of arrays for storing structured data, along with key operations like insertion, deletion, sorting, and searching. It also includes a formatted display of all records.The integration of a file system ensures data persistence even after the program is closed. This project highlights essential programming concepts such as structures, file handling, and menu-driven interfaces, making it a practical introduction to real-world data management in C.