Department of Mechanical, Industrial, and Mechatronics Engineering

Please select your current program below:

☐ **Mechanical Engineering**

☐ **Industrial Engineering**

☑ ~~**Mechatronics Engineering**~~

Course Number	MTE301
Course Title	Programming for Mechatronics Engineering
Semester/Year	Winter 2025
Section Number	4

Assignment No.3

Submission Date	Oct 17, 2025
Due Date	Oct 17, 2025

Student Name	Student ID (xxxx1234)	Signature*
Moosa Mughal	xxxx93125	

(Note: Remove the first 4 digits from your student ID)

Assignment 2 Problem 1

Database for films and songs.

1. Create separate classes for films & songs.
2. Note: Following steps are for both Film and Song classes
3. Add private member variables (name, director/singer, date) [All strings]
4. In public:
    - Create a default constructor and a constructor that sets the variables to the input.

```
Film/Song(string n, string d, string dat)
    {name = n; director/singer = d; date = dat;}
```

    - Create a getter function for name
    - Create an input friend member function for receiving input using getline.
        i. Parameters (inputstream type, and object Song/Film reference)
        ii.

```
friend istream &operator>>(istream &in, Film &f)//Line 30
```

    - Create an output friend member function for neatly outputting data.
        i.

```
friend ostream &operator<<(ostream &out, Song &s)
```

        ii. Kung Fu Panda 2 | Jennifer Yuh Nelson | 2011
5. Create a class **Record** for storing the objects and main database functions.(Line 87)
   #include <vector>
    - Create a private vector of type film and another vector of type song

```
vector<Film> films;
vector<Song> songs;
```

6. In public: (House main function)
7. **For showing records:**
    - Use a for loop and print all films/songs in vector in a numbered list.

```
        for (int i = 0; i < films.size(); i++)
        {
            cout << i + 1 << ". " << films[i] << endl;
        }
```

    - Uses ostream operator.
8. **For adding films/songs from console:**
    - Create temporary object of type film/song.
    - Use cin >> object to use the istream operator.
    - pushback object to appropriate vector.
9. **For removing films/songs:**
    - Display the records using a numbered list (showFilms() or showSongs()).
    - Ask which number to remove (with error handling).
    - Use .erase to erase that index.

```
songs.erase(songs.begin() + num - 1);
```

    - Print a message to confirm the object is removed from the vector.
10. **For sorting records:**
    - #include <algorithm>
    - Use sort function to sort the vector by name in ascending order.

```
sort(songs.begin(), songs.end(), [](Song &a, Song &b)
    { return a.getName() < b.getName(); });
```

- Note: need a lambda function to indicate how to sort as the objects are not a default type.
- Output a message that says the objects are sorted.

11. For outputting to file:

- Take string input for the film and song files.
- Declare those as the output files using ofstream.
- Using foreach loop output the objects to the correct file and close the files.
- Output saved records message.

12. For inputting from files

- Take string input for the film and song files.
- Declare those as the input files using ifstream.
- Create a temporary string for each line
- Use a while loop with getline to parse the files.
- Use .find() and .substr() to filter the formatting and only collect the data.

```
while (getline(songsIn, line))
{
    int p1 = line.find('|'), p2 = line.find('|', p1 + 1);
    if (p1 != string::npos && p2 != string::npos)
        songs.emplace_back(line.substr(0, p1 - 1), line.substr(p1
+ 2, p2 - p1 - 3), line.substr(p2 + 2));
}
```

- Use emplace_back to push the data to the vectors.

13. In main function()

- Create an object of type record (Record records;)
- Get data from files.

```
records.inFromFile("Films.txt", "Songs.txt");
```

- Use a while(true) loop for the menu
- Create a choice variable that takes console input and executes that task.
- Create a switch statement for easy execution.

```
 switch (choice)
{
case 0:
    cout << "\n\033[31m > Exiting...\033[0m";
    return 0;
case 1:
    records.showFilms();
    break;
case 2:
    records.showSongs();
    break;
```

```
        case 3:
            records.addFilm();
            break;
        case 4:
            records.addSong();
            break;
        case 5:
            records.removeFilm();
            break;
        case 6:
            records.removeSong();
            break;
        case 7:
            records.sortFilms();
            break;
        case 8:
            records.sortSongs();
        case 9:
            records.outToFile("Films.txt", "Songs.txt");
            break;
        default:
            cout << "\n\033[31m > Not Valid Choice\033[0m\n";
            break;
        }
```

- If choice is 0, use return to close the program.
- For default output that choice is not valid.

```
--- Actions Menu (Enter 0-9) ---
1. Display Films 2. Display Songs
3. Add Film      4. Add Song
5. Remove Film   6. Remove Song
7. Sort Films    8. Sort Songs
9. Save Records  0. Exit
Option: 1
1. Spirited Away | Hayao Miyazaki | 2001
2. Kung Fu Panda 2 | Jennifer Yuh Nelson | 2011
```

Assignment 2 Problem 2

Rectangles in an endless plane simulator.

1. Create a **class Rectangle**.
2. Add **private variables** (int left, top, width, height, right, bottom).
3. Add a **public constructor** that has parameters(left, top, width, height)
    - Set right to left + width, set bottom to top + height.
4. Create a **public void displayRect()** function.
    - Outputs to console the rectangle top left coordinate, and bottom right coordinates.
    - “Rectangle from (0, 0) to (5, 5)”
5. Create a **public void intersection()** function.
    - Takes in a reference to the other rect.
    - Create variables to find the coordinates of the intersection rectangle.
        i. Find max of the left using an algorithm function to get the edge closest to the right.
        ii. Find the max of the top to get the lowest top edge.
        iii. Find the min of right to find the edge closest to the left.
        iv. Find the min of bottom to find the edge closer to the top.
    - Check if rectangles overlap
        i. If (farthestRight < closestLeft && lowestTop < highestBottom) then create a rectangle with those coordinates
        ii. Display the intersection rectangle.
        iii. Else output that there's no intersection.
6. Create a **public void checkPoint()** function.
    - Takes in x and y point coordinates.
    - **If** the x cord:
        i. is more than or equal to left
        ii. and is less than or equal to right
    - **And if** the y cord:
        i. is more than or equal to top
        ii. and is less than or equal to bottom
    - Then the **point is in the** rectangle.
        i. Output: “Is inside Rectangle”
    - **Else:** Output: “Is Outside Rectangle”
7. In **main function**:
    - Create and display test rectangles
    - Test intersection()
    - Test checkPoint()