C programming Project

Submitted By: Saniya Singh

Sap Id: 590026316

Submitted To: Mr. Supreet Singh

BRANCH : SCHOOL OF COMPUTER SCIENCE

BATCH NUMBER: 51

YEAR: 2025_26

1. Project Title

MUSIC PLAYLIST MANAGER

Using C Programming

2. Introduction

The **Music Playlist Manager** is a console-based application created using the C programming language.
It allows users to manage a playlist by performing operations like **adding songs, deleting songs, viewing songs, searching for songs, and shuffling the playlist**.

This project demonstrates key C programming concepts such as structures, arrays, loops, string functions, and file handling.
It is designed to show how data can be organized and stored efficiently using basic programming logic.

3. Objective of the Project

The main purpose of this project is to simulate how real applications like Spotify or Wynk manage songs, but in a simple console form.

The goals of the project are:

- To store and manage song information efficiently.
- To allow users to interact with the playlist through a simple menu-driven program.
- To apply important C concepts in a practical project.
- To create a system where song data is saved permanently using files.

---

4. Features of the System

1. Add Song:

The user enters-

- Song Title
- Artist Name
- Genre

The program stores this in a structure array.

2. View All Songs:

Shows all songs in the playlist in an organized format.

3. Delete a Song:

The user selects the song number, and the program removes it by shifting elements.

4. Search Song:

Searches for a song using **case-insensitive comparison**.
If found, it displays all details of the song.

5. Shuffle Songs:

Randomly rearranges the playlist using:
“rand() % count”

This simulates how music apps shuffle songs.

6. Save Playlist to File:

The playlist is saved into a text file:
“playlist.txt”.

This helps the data stay even after the program closes.

7. Load Playlist on Startup:

When the program starts, it automatically loads saved songs from the file.

5. System Design

Flow of the Program:

1. Load playlist from file
2. Show menu
3. Ask user for choice
4. Perform operation:
    - Add Song
    - View Songs
    - Delete Song
    - Search Song

- Shuffle
- Save

5. Repeat until user exits

Menu-Driven Approach:

The user interacts with the system through a repeating menu, making the application simple and user-friendly.

6.Explanation of Concepts Used

1. Structures:

A structure is used to store multiple data types together.
We created a structure for each song:

```
struct Song {
   char title[50];
   char artist[50];
   char genre[30];
};
```

This allows storing all song details in one unit.

---

2. Arrays of Structures:

To store multiple songs:

"struct Song playlist[100];"

This means we can store up to 100 songs.

---

3. String Handling:

I used:

- strcmp() / stricmp() → for searching
- scanf() with " %[^\n]" → to take full line input including spaces
- strcpy() → for copying text

---

4. Loops and Conditional Statements:

Used to:

- Navigate menu
- Add/delete songs
- Display playlist
- Perform search and shuffle

These control the program flow.

---

5. Randomization (Shuffle Feature):

We use:

```
srand(time(0));
int r = rand() % count;
```

This swaps random songs to shuffle the playlist.

---

6. File Handling:

To save and load the playlist, we used:

Function	Purpose
fopen()	open file
fprintf()	write song data
fscanf()	read song data

Function Purpose

fclose() close file

This gives **permanent storage**, which is essential for any real application.

7.Advantages

- Easy to understand and operate
- Demonstrates real-life application logic
- Uses file handling for data storage
- Can be expanded into a bigger project
- Lightweight and fast

8.Limitations

- Only text-based (no GUI)
- Cannot play actual music files
- Playlist size has a fixed limit
- Does not support multiple user accounts

9.CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

struct Song {
    char title[50];
    char artist[50];
    char genre[30];
};

struct Song playlist[100];
int count = 0;

void addSong() {
    printf("Enter Title: ");
    scanf(" %[^\n]", playlist[count].title);

    printf("Enter Artist: ");
    scanf(" %[^\n]", playlist[count].artist);

    printf("Enter Genre: ");
    scanf(" %[^\n]", playlist[count].genre);

    count++;
    printf("Song Added Successfully!\n");
}

void viewSongs() {
    if (count == 0) {
        printf("\nNo songs in playlist.\n");
        return;
    }

    printf("\n------ PLAYLIST ------\n");
    for (int i = 0; i < count; i++) {
        printf("%d. %s | %s | %s\n", i + 1, playlist[i].title, playlist[i].artist, playlist[i].genre);
```

```
    }
}

void deleteSong() {
    viewSongs();
    if (count == 0) return;

    int index;
    printf("\nEnter song number to delete: ");
    scanf("%d", &index);

    if (index < 1 || index > count) {
        printf("Invalid choice.\n");
        return;
    }

    for (int i = index - 1; i < count - 1; i++) {
        playlist[i] = playlist[i + 1];
    }

    count--;
    printf("Song Deleted Successfully!\n");
}

void searchSong() {
    char key[50];
    printf("Enter song name to search: ");
    scanf(" %[^\n]", key);

    for (int i = 0; i < count; i++) {
        if (stricmp(playlist[i].title, key) == 0) {
            printf("\nFound: %s | %s | %s\n", playlist[i].title, playlist[i].artist, playlist[i].genre);
            return;
        }
    }
    printf("Song not found.\n");
}
```

```
void shuffleSongs() {
    if (count < 2) {
        printf("Not enough songs to shuffle.\n");
        return;
    }

    srand(time(0));
    for (int i = 0; i < count; i++) {
        int r = rand() % count;
        struct Song temp = playlist[i];
        playlist[i] = playlist[r];
        playlist[r] = temp;
    }
    printf("Playlist Shuffled Successfully!\n");
}

void savePlaylist() {
    FILE *fp = fopen("playlist.txt", "w");

    for (int i = 0; i < count; i++) {
        fprintf(fp, "%s|%s|%s\n", playlist[i].title, playlist[i].artist, playlist[i].genre);
    }

    fclose(fp);
    printf("Playlist Saved!\n");
}

void loadPlaylist() {
    FILE *fp = fopen("playlist.txt", "r");
    if (!fp) return;

    count = 0;
    while (fscanf(fp, " %[^|]|%[^|]|%[^\n]\n",
                  playlist[count].title,
                  playlist[count].artist,
                  playlist[count].genre) != EOF) {
```

```
        count++;
    }

    fclose(fp);
}

int main() {
    loadPlaylist();
    int choice;

    while (1) {
        printf("\n--- MUSIC PLAYLIST MANAGER ---\n");
        printf("1. Add Song\n");
        printf("2. View Songs\n");
        printf("3. Delete Song\n");
        printf("4. Search Song\n");
        printf("5. Shuffle Playlist\n");
        printf("6. Save Playlist\n");
        printf("7. Exit\n");
        printf("Choose: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addSong(); break;
            case 2: viewSongs(); break;
            case 3: deleteSong(); break;
            case 4: searchSong(); break;
            case 5: shuffleSongs(); break;
            case 6: savePlaylist(); break;
            case 7: printf("Goodbye!\n"); return 0;
            default: printf("Invalid choice.\n");
        }
    }
}
```

10. OUTPUT EXPLANATION:

The output of the **Music Playlist Manager** is entirely menu-driven.
When the program runs, the user is shown a list of options.
Based on what the user chooses, different outputs appear.
Here is the explanation of each output:

---

1. **Program Start-**

When the program begins, it automatically loads any previously saved songs from the file playlist.txt.

Output Example:

This is the **main menu**, and it repeats after every action.

---

2. **Adding a Song-**

When the user chooses **Option 1: Add Song**, the program asks for song details.

Output:

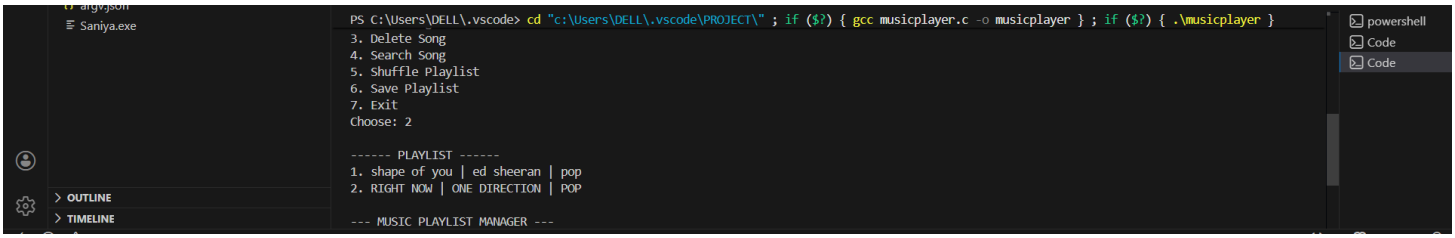
The song is stored in the playlist array and increases the total number of songs.

---

3. Viewing All Songs-

When the user chooses **Option 2**, all songs are displayed one by one with numbering.

Output:



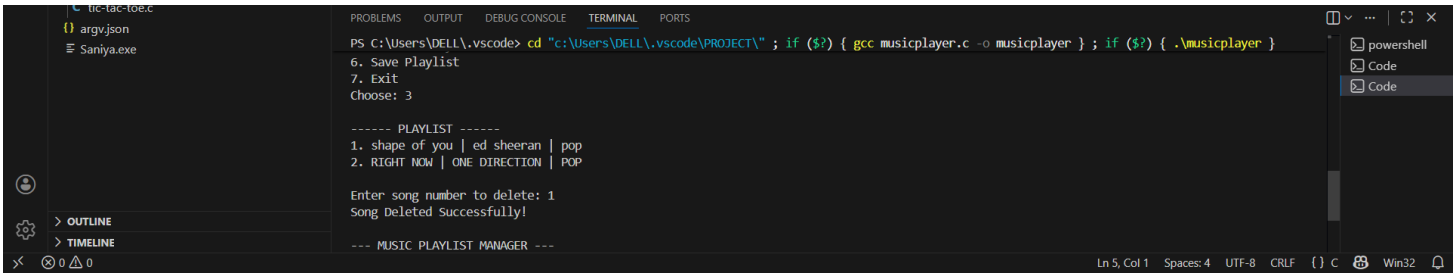
Each song's **title**, **artist**, and **genre** appear in a clean format.

---

4. Deleting a Song-

When **Option 3** is chosen, the program first shows the playlist and then asks for the song number to delete.

Output:



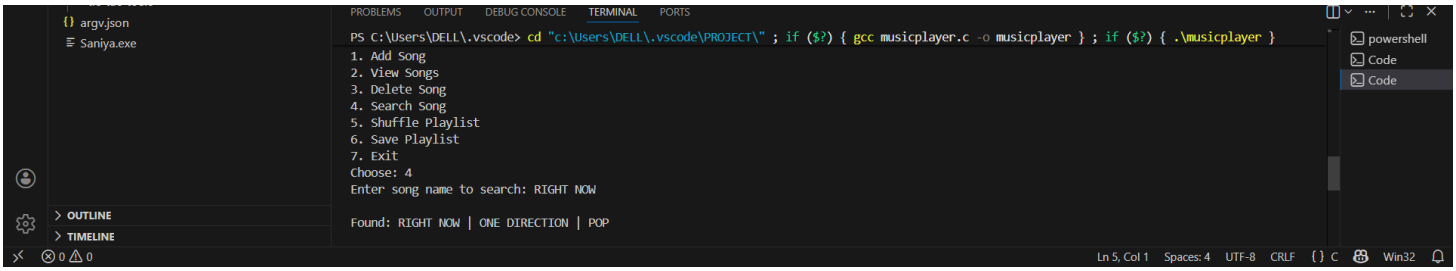
The program removes the selected song and shifts the rest upward.

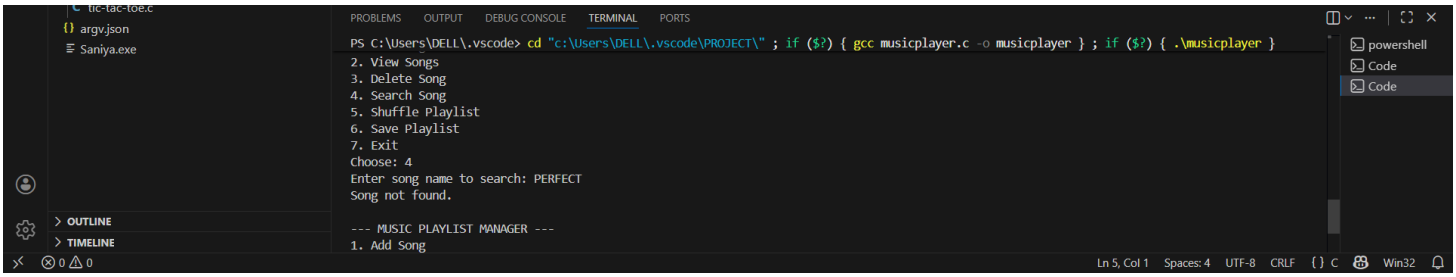
---

5. Searching for a Song-

When **Option 4** is selected, the program searches for the title entered by the user.

Output:





The program compares the input with each title using **case-insensitive** matching.

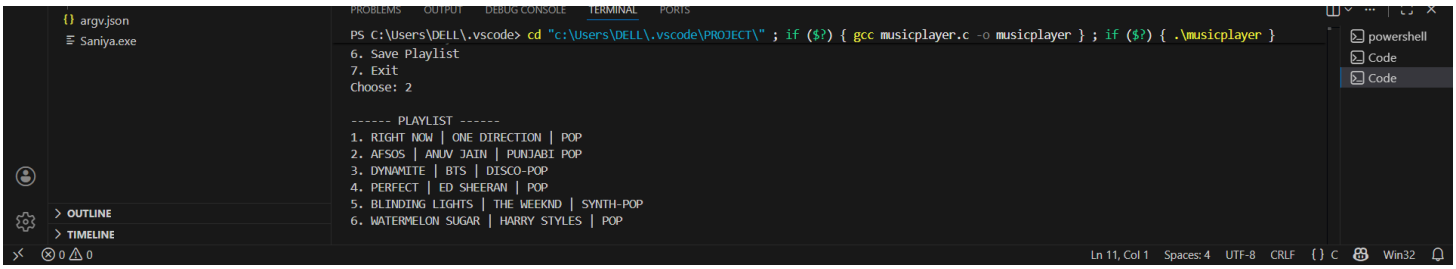
---

6. Shuffling the Playlist-

When **Option 5** is selected, the playlist order is randomly changed.

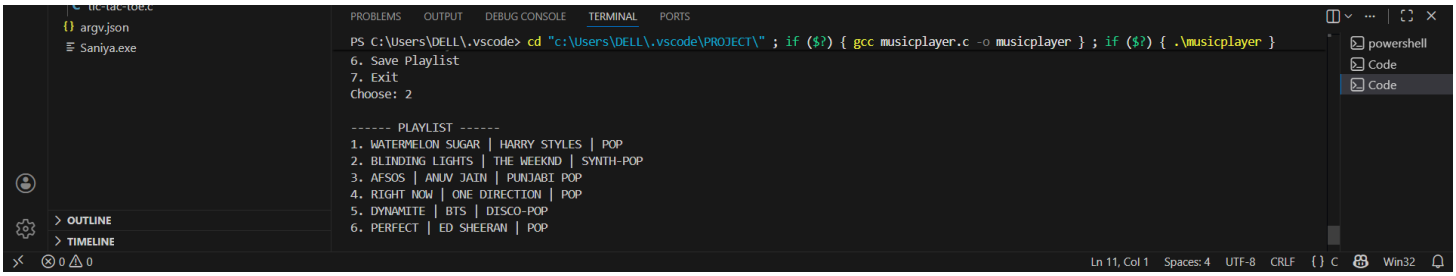
Output:

BEFORE SHUFFLING-



SHUFFLING DONE-

AFTER SHUFFLING-



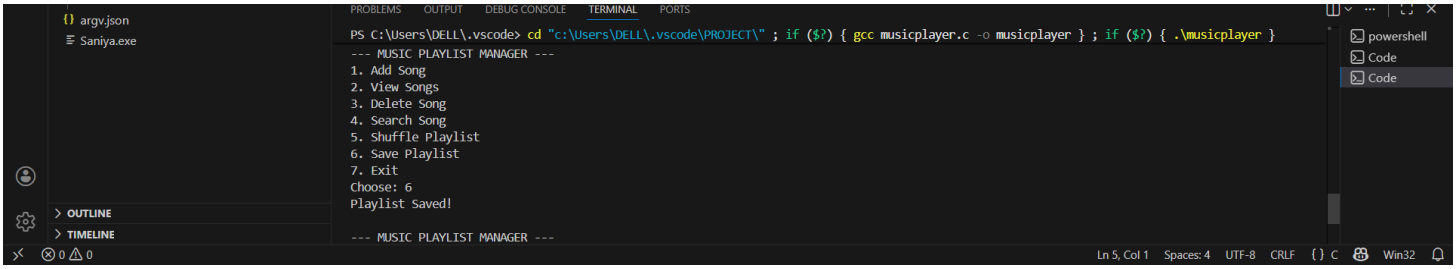
The songs are swapped randomly using the rand() function.

---

7. Saving the Playlist-

When the user chooses **Option 6**, all songs are written into a file.

Output:



The file playlist.txt is updated so the playlist stays even after closing the program.

---

8. Exiting the Program-

Choosing **Option 7** ends the program.

Output:

The loop stops, and the program terminates safely.

Summary of Output Behavior:

Operation	What you see in output	Why it happens
1.Add	Prompts for details → success message	New song stored
2.View	List of all songs	Reads array and prints it
3.Delete	Shows playlist → deletes chosen song	Shifts elements
4.Search	Either "Found" or "Not found"	String comparison
5.Shuffle	"Shuffled Successfully"	Random swaps
6.Save	"Playlist Saved"	Writes to file
7.Exit	"Goodbye!"	Ends program

11. Conclusion

The **Music Playlist Manager** is a functional and educational C project. It demonstrates how data can be stored, managed, and saved using structures, arrays, and file handling.

This project helps build logical thinking and gives practical experience with C programming in a real-world scenario.