MUSIC PLAYLIST ARRANGER

NAME- PRASIDDHI ROY

SAP ID- 590021861

BATCH-37

SUBJECT- PROGRAMMING IN C

ABSTRACT

The MusicPlaylistArranger is a command-line C program designed to help users manage and organize music playlists efficiently. The program allows users to add new songs, search for songs, display the entire playlist, and save or load playlists from a file. Using a modular design with separate header and source files, the project demonstrates fundamental programming concepts such as arrays, file handling, and string manipulation in C.

This project provides a simple yet effective interface for managing music collections, emphasizing usability, data persistence, and scalability. While the current version operates entirely via the terminal, the structure allows for future enhancements such as song deletion, editing, or even integration with audio playback libraries. Overall, MusicPlaylistArranger serves as a practical exercise in programming logic, file management, and user interaction design.

PROBLEM DEFINITION

Managing a growing collection of music manually can be hectic and inefficient, especially when trying to find, organize, or store songs systematically. Users often struggle with keeping track of their favorite tracks, searching for specific songs quickly, and maintaining a persistent list across multiple sessions.

The MusicPlaylistArranger project addresses this problem by providing a simple, terminal-based program that allows users to add songs, search for specific songs, display the entire playlist, and save/load playlists from a file. The goal is to provide an organized, persistent, and user-friendly system for managing music collections, while demonstrating fundamental programming concepts such as arrays, file handling, and modular design in C.

SYSTEM DESIGN

*ALGORITHM (MAIN.C)

1. Start
2. Load songs by calling loadSongs()
3. Show menu and get user choice
4. Based on choice:
   - Add Songs → addSongs()
   - Search Songs → searchSongs()
   - Display Songs → displaySongs()
   - Save Songs → saveSongs()
   - Exit → save and terminate
5. Repeat menu until user exits
6. stop

ALGORITHM(playlist.c)

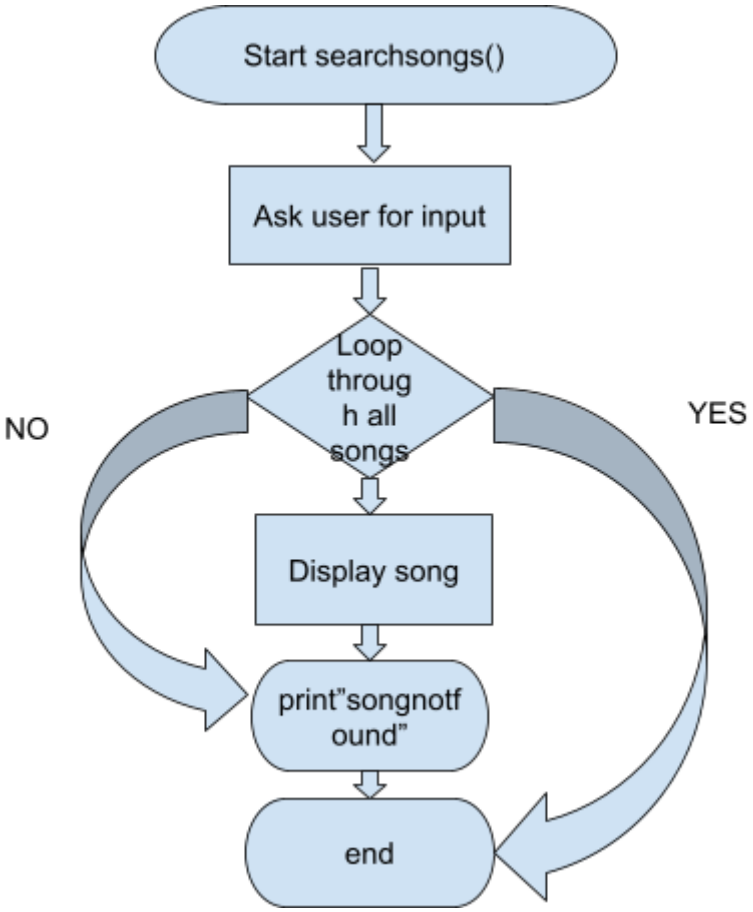
1. Start
2. Declare global array songs and variable songCount
3. Declare function prototypes:
4. addSongs()
   searchSongs()
   displaySongs()
   saveSongs()
   loadSongs()
5. Stop

ALGORITHM(playlist.h)

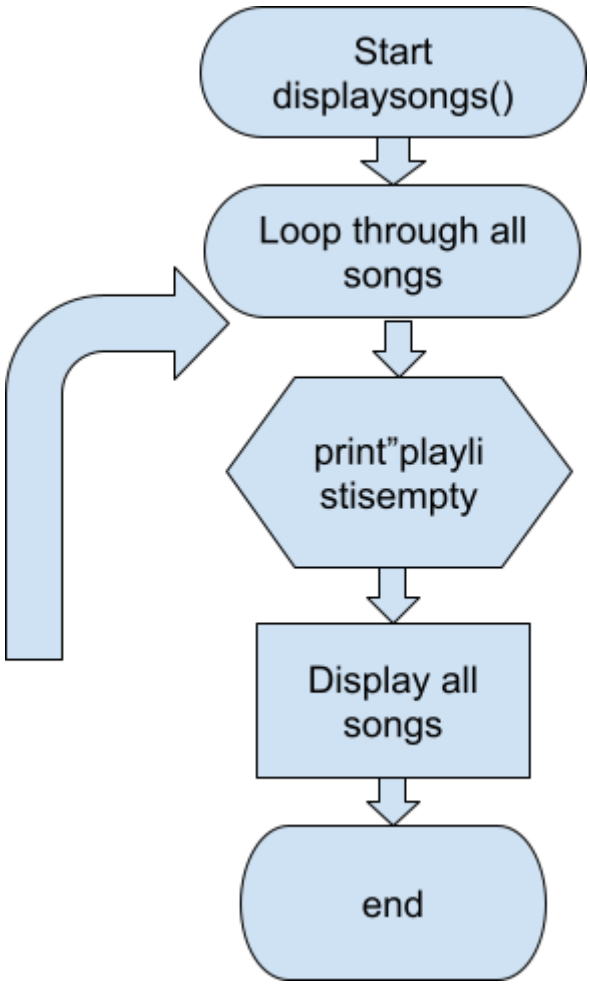
1. Start
2. Declare global array songs and variable songCount
3. Declare function prototypes:
4. addSongs()
   searchSongs(),
   displaySongs(),
   saveSongs(),
   loadSongs()
5. Stop

FLOWCHART

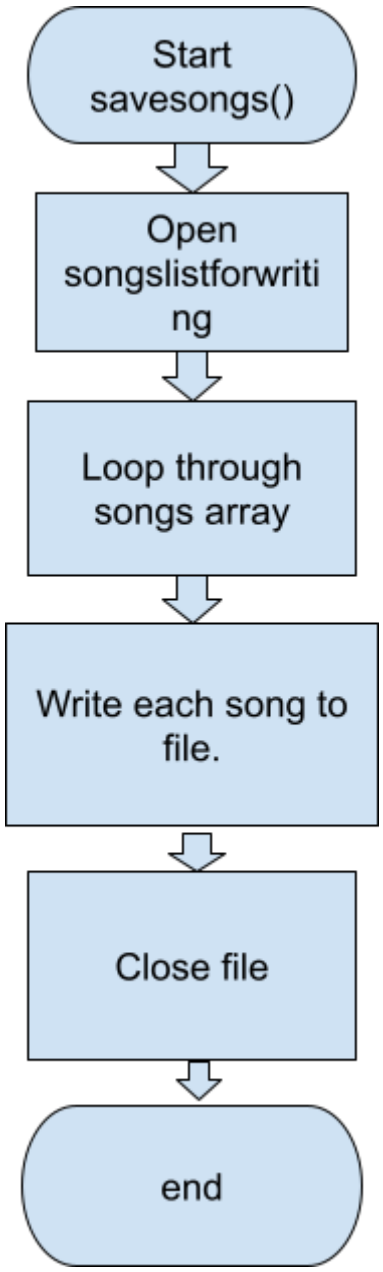
1. searchsongs()



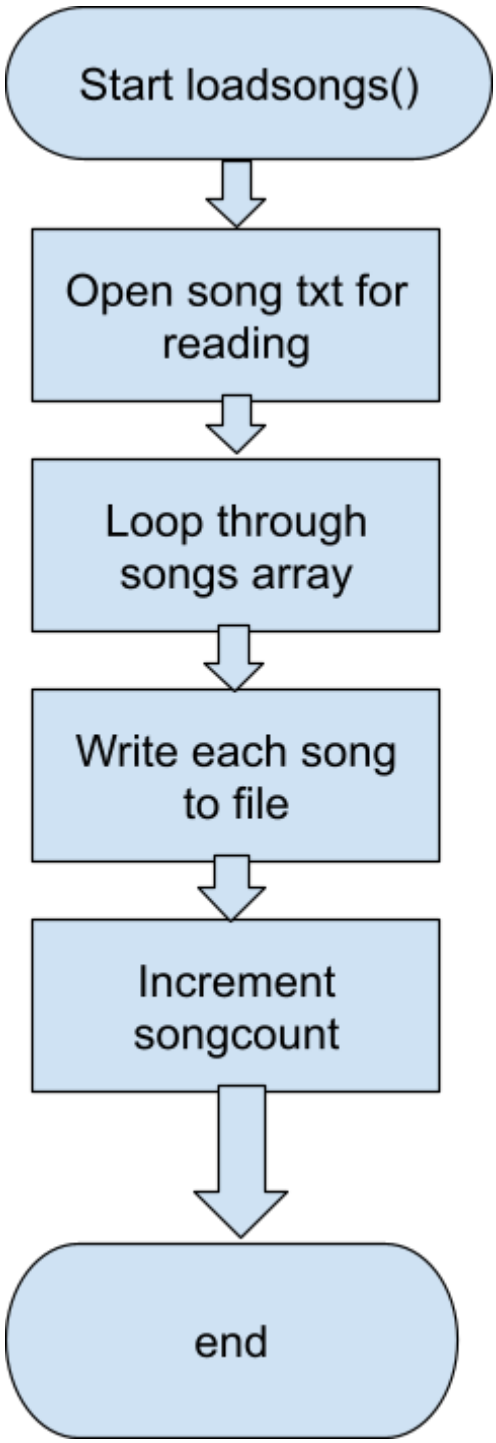
2. Displaysongs()



3. savesongs()


Start savesongs()
Open songslistforwriting
Loop through songs array
Write each song to file.
Close file
end

4. loadsongs()


Start loadsongs()
Open song txt for reading
Loop through songs array
Write each song to file
Increment songcount
end

IMPLEMENTATION DETAILS

It will tell how is divide-

1> **structure-**
Program is divided into 3 parts-
-**main.c-** contains all thecore of the program and calling of functions etc.
-**playlist.c-**contains logic of the program
-playlist.h-a header file where all the functions has been declared.

2>addsongs()

```c
void addsongs()
{
int n , added = 0;
printf("enter the number of songs u want to add");
scanf("%d", &n);
getchar(); //consume newline
for (int i=0; i<n; i++)
{
    if(countsongs>=50)
    {
        printf("no space for new songs\n");
        break;
    }
printf("enter song %d:" , i+1);// we want song to store from 1
fgets(songs[countsongs], 50 , stdin); /*songs store count songs
stdin make sure fgets read the input from keyboard
50= max characters*/
songs[countsongs][strcspn(songs[countsongs], "\n")]=0;
/* in strcpsn line it gives the index of where /n is
then songs[countsongs][index]=0 replaces /n with null operator*/
countsongs++;
added++;
}
printf("songs added successfullly are:\n", added);
}
```

3>searchthesongs()

```c
// search song by name
void searchthesongs()
{
    char name[50];
    printf("enter the song to search :");
    getchar();
    fgets(name , 50, stdin);
    name[strcspn(name, "\n")]=0;
    int found=0;
    for( int i=0 ; i<countsongs ; i++)
    {
        if(strcmp(songs[i], name)==0) // ==0 means that they are exactly equal
        {
            printf("song found at the position %d\n", i+1);
            found = 1;
            break;

        }
    }
    if (!found) printf("song not found\n");
}
// load songs from songs.txt
```

4>loadthesongs()

```c
void loadsongsfromfile()
{
    FILE *fp = fopen("../data/songs.txt", "r"); /* fopen opens the file
    r is read mode as we don't want to write the song
    fp file pointer use to access the file*/
    if(fp == NULL) {
        printf("Could not open songs.txt. Starting with empty playlist.\n");
        countsongs = 0;
        return;
    }
    countsongs = 0; //reset
    while(fgets(songs[countsongs], 50, fp) != NULL) // stdin and fp both are streamers but stdin take inputs from keyboard but fp from that particular file
    {
        songs[countsongs][strcspn(songs[countsongs], "\n")] = 0; // remove newline
        countsongs++;
    }
    fclose(fp);// close file after done using to prevent memory leakage
}
```

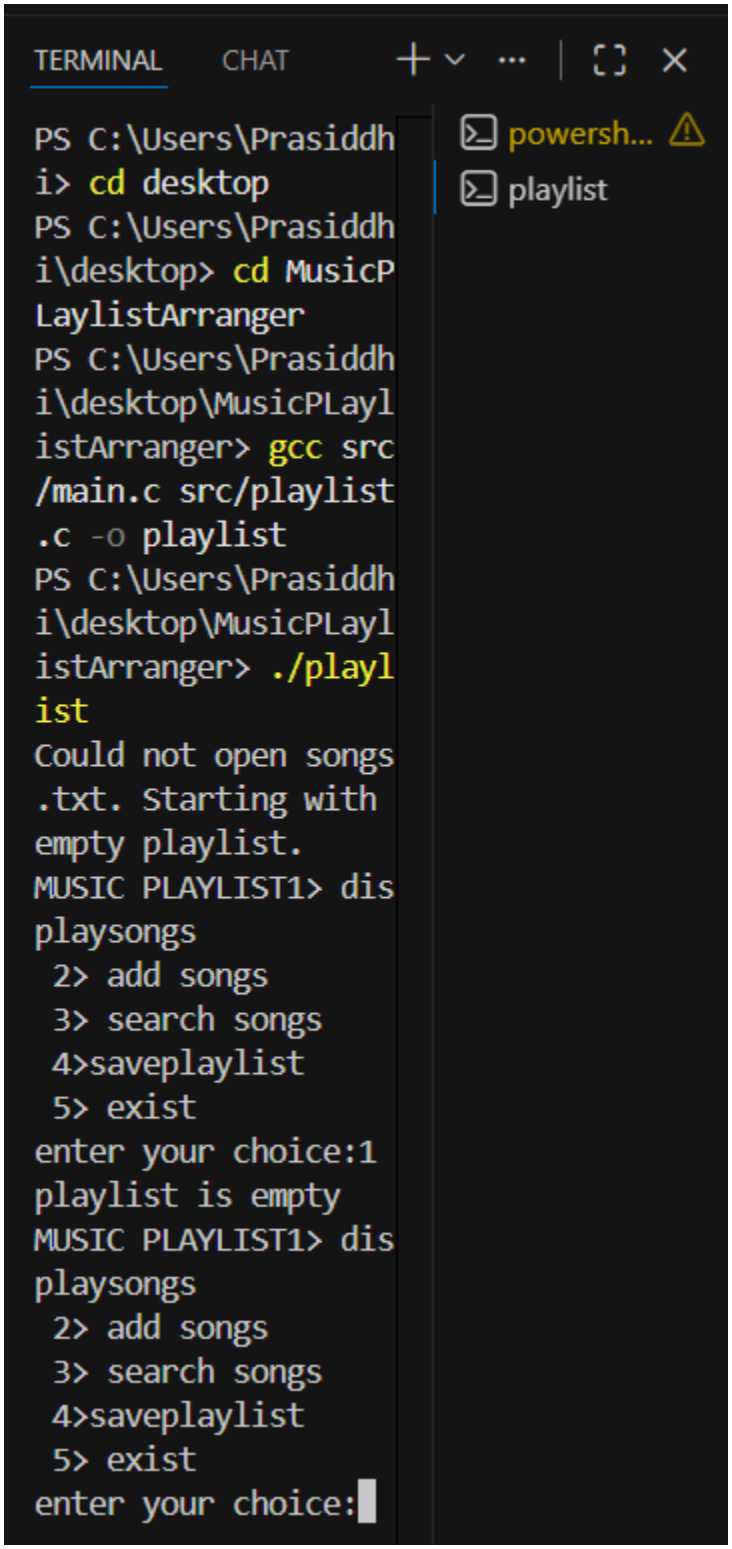
5>savethesongs()

```c
void savesongstofile() // it rewrites songs which were loaded and which were added by removing it all
{
    FILE *fp = fopen("../data/songs.txt", "w");
    if(fp == NULL) {
        printf("Could not save songs.\n");
        return;
    }
    for(int i = 0; i < countsongs; i++) {
        fprintf(fp, "%s\n", songs[i]); /* fprintf(file_pointer,"format string", values...)
        fpfile where we write
        song[i] string we want to write
        fprintf prints outputs to file not us*/
    }
    fclose(fp);
    printf("Playlist saved successfully.\n");
}
```

6>showsongs()

```c
void showsongs()
{
    if(countsongs==0)
    {
        printf("playlist is empty\n");
        return;
    }
    printf("PLAYLIST");
    for (int i=0; i<countsongs; i++)
    {
        printf("%d. %s\n", i+1, songs[i]);
    }
}
```

TESTING AND RESULT



CONCLUSION

The Music Playlist Arranger project successfully demonstrates the implementation of a basic music management system using C programming. Users can add, search, and display songs in a playlist, with all data stored persistently in a text file. The project highlights modular programming concepts, file handling, and user interaction via a menu-driven interface. Overall, it provides a simple yet effective solution for organizing and managing songs digitally.

FUTURE WORKS

This project can be extended and enhanced in several ways:

1. Playback functionality: Integrate audio playback for songs.
2. Playlist categories: Allow creation of multiple playlists (e.g., genres, moods).
3. Song deletion & editing: Enable users to remove or update song details.
4. Sorting & filtering: Sort songs alphabetically, by date added, or by user preference.

REFERNCES

1> LET US C
2>. GeeksforGeeks, C File Handling, https://www.geeksforgeeks.org/c-file-handling/
3>CLASS PPT
4>GIT HUB