PROJECT TITLE : MUSIC PLAYLIST IN C

INTRODUCTION:

This project is focused on customers selected playlist on their apps so it is named as “MUSIC PLAYLIST” .

Here, you can create a new playlist, insert new songs as and then, delete them and also print them.

Overall, with this project we can experience a real life music playlist. Music playlist in C is a console application without graphics.

Functions used in music playlist:

The source code for Customer Music Playlist is relatively short and easy to understand. I have divided this C mini project into many functions, most of which are related to all the options in the playlist. Listed below are some of the more important functions which may help you understand the project better.

Input String From User () Gets input from user,

Song Name Duplicate () Prints a message when a duplicate of the song is found in the library.

Song Name Found () Prints a message when a song name was found in the personal music library.

Song Name Not Found () Prints a message when a song name was not found in the personal music library.

Song Name Deleted () Prints a message when a song was deleted in the personal music library.

Artist Found () Prints a message when a artist name was found in the personal music library.

Artist Not Found () Prints a message when a artist name was not found in the personal music library

Print Music Library Empty () Prints a message when the personal music library is empty.

Print Music Library Title () Prints the title of the music library.

print Music Library List () / Prints the music library (P Command)

SOURCE CODE:

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include <stdbool.h>

typedef struct node {

char \*artist;

char \*songName;

char \*genre;

struct node \*linkToNextNode;

} Node;

const int MAX\_LENGTH = 1024;

void inputStringFromUser(char \*prompt, char \*s, int arraySize);

void songNameDuplicate(char \*songName); void songNameFound(char \*songName); void songNameNotFound(char \*songName); void songNameDeleted(char \*songNamevoid artistFound(char \*artist);

void artistNotFound(char \*artist

void printMusicLibraryEmpty(void);

void printMusicLibraryTitle(void);

void printMusicLibraryList(Node \*head);

void searchForSong(Node \*head, char songIntededToBeFound[MAX\_LENGTH]);

bool songAlreadyInPlaylist(Node \*head, char songIntededToBeFound[MAX\_LENGTH]);

Node \*insertANewNodeInOrder(Node \*head, char songName[MAX\_LENGTH], char artist[MAX\_LENGTH], char genre[MAX\_LENGTH]);

Node \*deleteTheSong(Node \*head, char songIntededToBeDeleted[MAX\_LENGTH]);

Node \*deleteAllNodesFromLibrary(Node \*head);

int main(void)

{

Node \*head = NULL;

char \*songName, \*artist, \*genre;

artist = (char \*) malloc(MAX\_LENGTH \* sizeof (char));

genre = (char \*) malloc(MAX\_LENGTH \* sizeof (char));

songName = (char\*) malloc(MAX\_LENGTH \* sizeof (char));

printf("Personal Music Library.\n\n");

printf("%s", "Commands are I (insert), D (delete), S (search by song name),\n"

"P (print), Q (quit).\n");

char response;

char input[MAX\_LENGTH + 1];

do

{

inputStringFromUser("\nCommand", input, MAX\_LENGTH); character entered by user.

response = toupper(input[0]);

if (response == 'I') {

char \*promptName = "Song name";

char \*promptArtist = "Artist";

char \*promptGenre = "Genre";

inputStringFromUser(promptName, songName, MAX\_LENGTH);

inputStringFromUser(promptArtist, artist, MAX\_LENGTH);

inputStringFromUser(promptGenre, genre, MAX\_LENGTH);

if (!songAlreadyInPlaylist(head, songName))

{

head = insertANewNodeInOrder(head, songName, artist, genre);

}

else

{

songNameDuplicate(songName);

}

}

else if (response == 'D')

{

char \*prompt = "\nEnter the name of the song to be deleted";

inputStringFromUser(prompt, songName, MAX\_LENGTH);

head = deleteTheSong(head, songName);

}

else if (response == 'S')

{

char \*prompt = "\nEnter the name of the song to search for";

inputStringFromUser(prompt, songName, MAX\_LENGTH);

searchForSong(head, songName);

}

else if (response == 'P') {

printMusicLibraryList(head);

}

else if (response == 'Q')

{

;

}

else

{

printf ("\nInvalid command.\n");

}

} while (response != 'Q'); // Delete the entire linked list.

head = deleteAllNodesFromLibrary(head);

printMusicLibraryList(head); // Print the linked list to confirm deletion.

return 0;

}

void inputStringFromUser(char \*prompt, char \*s, int maxStrLength)

{

int i = 0;

char c;

printf("%s --> ", prompt);

while (i < maxStrLength && (c = getchar()) != '\n')

s[i++] = c;

s[i] = '\0';

}

void songNameDuplicate(char \*songName)

{

printf("\nA song with the name '%s' is already in the music library.\n"

"No new song entered.\n", songName);

}

void songNameFound(char \*songName)

{

printf("\nThe song name '%s' was found in the music library.\n\n", songName);

}

void songNameNotFound(char \*songName)

{

printf("\nThe song name '%s' was not found in the music library.\n", songName);

}

void songNameDeleted(char \*songName)

{

printf("\nDeleting a song with name '%s' from the music library.\n", songName);

}

void printMusicLibraryEmpty(void) {

printf("\nThe music library is empty.\n");

}

void printMusicLibraryTitle(void)

{

printf("\nMy Personal Music Library: \n");

}

void printMusicLibraryList(Node \*head)

{

Node \*current = head;

if (current == NULL)

{

printMusicLibraryEmpty();

}

else

{

printMusicLibraryTitle();

while (current != NULL)

{

printf("\n");

printf("%s\n", current -> songName);

printf("%s\n", current -> artist);

printf("%s\n", current -> genre);

current = current -> linkToNextNode;

}

}

}

void searchForSong(Node \*head, char songIntededToBeFound[MAX\_LENGTH])

{

if (head == NULL)

{

return;

}

Node \*current = head;

while (current != NULL)

{

if (strcmp(current -> songName, songIntededToBeFound) == 0)

{

songNameFound(songIntededToBeFound);

printf("%s\n", current -> songName);

printf("%s\n", current -> artist);

printf("%s\n", current -> genre);

return;

}

current = current -> linkToNextNode;

}

songNameNotFound(songIntededToBeFound);

return;

}

bool songAlreadyInPlaylist(Node \*head, char songIntededToBeFound[MAX\_LENGTH])

{

if (head == NULL)

{

return false;

}

Node \*current = head;

while (current != NULL)

{

if (strcmp(current->songName, songIntededToBeFound) == 0)

{

return true;

}

current = current -> linkToNextNode;

}

return false;

}

// This is the function for inserting a new node (the I command) into the linked list.

Node \*insertANewNodeInOrder(Node \*head, char songName[MAX\_LENGTH], char artist[MAX\_LENGTH], char genre[MAX\_LENGTH])

{

Node \*linkToNextNode = head;

if (head == NULL || strcmp(head -> songName, songName) > 0)

{

Node \*insert = (Node \*) malloc( sizeof(Node));

insert -> songName = (char \*) malloc(MAX\_LENGTH \* sizeof(char));

insert -> artist = (char \*) malloc(MAX\_LENGTH \* sizeof(char));

insert -> genre = (char \*) malloc(MAX\_LENGTH \* sizeof(char));

if (insert != NULL)

{

strcpy(insert -> songName, songName);

strcpy(insert -> artist, artist);

strcpy(insert -> genre, genre);

insert -> linkToNextNode = linkToNextNode;

}

return insert;

}

Node \*current = head;

while (current -> linkToNextNode != NULL && strcmp(current -> linkToNextNode -> songName, songName) < 0)

{

current = current -> linkToNextNode;

}

Node \*insert = (Node \*) malloc(sizeof (Node));

insert -> songName = (char \*) malloc(MAX\_LENGTH \* sizeof (char));

insert -> artist = (char \*) malloc(MAX\_LENGTH \* sizeof (char));

insert -> genre = (char \*) malloc(MAX\_LENGTH \* sizeof (char));

if (insert != NULL)

{

strcpy(insert -> songName, songName);

strcpy(insert -> artist, artist);

strcpy(insert -> genre, genre);

insert -> linkToNextNode = linkToNextNode;

}

current -> linkToNextNode = insert;

return head;

}

Node \*deleteTheSong(Node \*head, char songIntendedToBeDeleted[MAX\_LENGTH])

{

if (head == NULL)

{

return NULL;

}

if (strcmp(head -> songName, songIntendedToBeDeleted) == 0)

{

Node \*secondNode = head -> linkToNextNode;

songNameDeleted(head -> songName);

free(head -> songName);

free(head -> artist);

free(head -> genre);

free(head);

return secondNode;

}

bool foundTheSong = false;

Node \*current = head;

while (!foundTheSong && current -> linkToNextNode != NULL)

{

if (strcmp(current -> linkToNextNode -> songName, songIntendedToBeDeleted) == 0)

{

foundTheSong = true;

}

else

{

current = current -> linkToNextNode;

}

}

if (current -> linkToNextNode != NULL)

{

Node \*songToRemove = current -> linkToNextNode;

current -> linkToNextNode = current -> linkToNextNode -> linkToNextNode;

songNameDeleted(songToRemove -> songName);

free(songToRemove -> songName);

free(songToRemove -> artist);

free(songToRemove -> genre);

free(songToRemove);

}

return head;

}

Node \*deleteAllNodesFromLibrary(Node \*head)

{

if (head == NULL)

{

return head;

}

while (head != NULL)

{

Node \*firstNode = head;

head = head -> linkToNextNode;

songNameDeleted(firstNode -> songName);

free(firstNode -> songName);

free(firstNode -> artist);

free(firstNode -> genre);

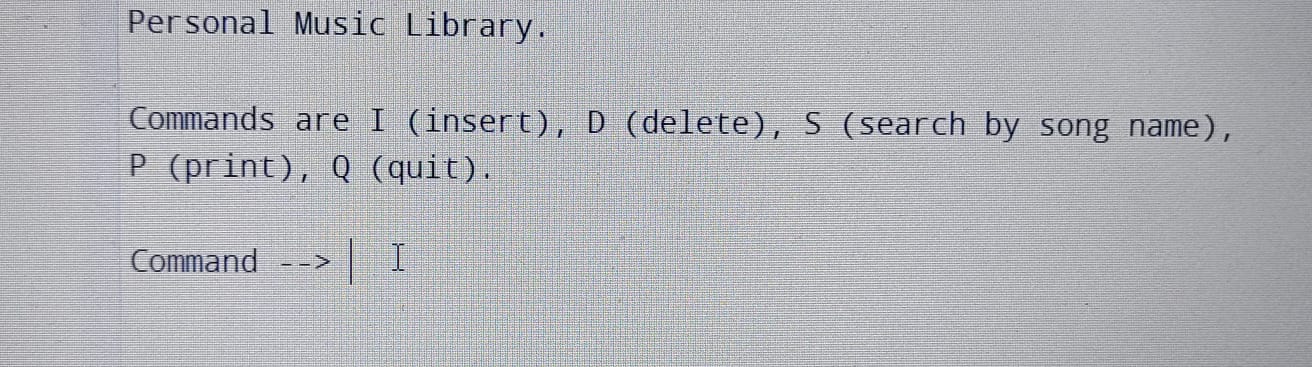
free(firstNode);

}

return head;

}

OUTPUT SCREENSHOTS:



CONCLUSIONS:

After finishing the music playlist system using the c programming language, it is clear that this system is a useful too for accessing our favourite music playlist, adding songs, deleting songs and also finding the author. This project shows how powerful and flexible the c language is, as well as how it can handle tasks in a clear and efficient way. Overall, putting the music playlist system into place using C was a success, and it is expected to be a useful toll for easily using the playlist. s