#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

#include <random>

#include <algorithm>

#include <vector>

#include <cstdio>

using namespace std;

struct COLLECTION { //declaration of structure variable

string title;

string album;

string artist;

string composer;

string genre;

string releaseDate;

string directory;

};

int readFile(struct COLLECTION songList[]); //read text file

int menu(); //show menu

void addSong();

void deleteSong(struct COLLECTION musicList[], int count ); //delete song

void updateSong();

void listSong(struct COLLECTION musicList[], int count); //list songs

void searchSong(struct COLLECTION musicList[], int count); // search song

void genPlaylist(struct COLLECTION musicList[], int count); //generate playlist

void exitApp(); //close text file

int main()

{

const int SIZE = 2000; //array of size 2000

struct COLLECTION songList[SIZE]; //declare array of structure

int choice, num; //declare choice and num (rows in text file)

char yn;

num = readFile(songList); //call function readFile and return int num

do { //do this while user decision (yn) is y that is return menu

choice = menu(); //call function menu and return int choice

cout << endl; //for a tidy program purpose

do { //do this while user decision (yn) is n that is continue same operation

switch (choice) { //directing to menu of choice

case 1: addSong(songList, num); num += 1; break; //num increased by 1 after add songs function in call

case 2: deleteSong(songList, num);num += 1; break;

case 3: editSong(songList, num); break;

case 4: listSong(songList, num); break;

case 5: searchSong (songList, num);num += 1; break;

case 6: genPlaylist(songList, num); break;

case 7: exitApp(songList, num); break;

}

do { //data validation to ask user whether to return menu or continue same operation

cout << "\nEnter Y if you wish to return to menu or N if you wish to continue this operation: ";

cin >> yn;

if (tolower(yn) != 'y' && tolower(yn) != 'n')

cout << "ERROR: PLEASE ENTER Y OR N ONLY" << endl;

if (choice == 7 && tolower(yn) == 'n') { //if user choice is 7 (exitApp) and continue operation (yn = n), end of program

cout << "\n\t\t\t\t--- Closing App ---\n\n\t\t\t\t HAVE A GOOD DAY!:)" << endl;

return 0;

}

} while (tolower(yn) != 'y' && tolower(yn) != 'n');

} while (tolower(yn) == 'n');

} while (tolower(yn) == 'y');

}

# int readFile(struct COLLECTION songList[]) //readFile function by sending in structure as parameter

{

ifstream inputFile; //declare input stream for file

inputFile.open("songlist.txt"); //connect a stream to a file named "songlist.txt"

int count = 0; //initialise count (rows in textfile)

string line, title, album, artist, composer, genre, releaseDate, directory; //declare variables

if (inputFile.is\_open()) { //if file is open,

getline(inputFile, line); //get the first line that we do not want

while (!inputFile.eof()) { //while file has not reach eof, assign value into

musicList[count];

getline(inputFile, musicList[count].title, ','); //get title until ','

getline(inputFile, musicList[count].album, ','); //get album until ','

getline(inputFile, musicList[count].artist, ','); //get srtist until ','

getline(inputFile, musicList[count].composer, ','); //get composer until ','

getline(inputFile, musicList[count].genre, ','); //get genre until ','

getline(inputFile, musicList[count].releaseDate, ','); //get releaseDate until ','

getline(inputFile, musicList[count].directory, '\n'); //get directory until '\n'

count++; //increase count by 1

}

inputFile.close();

}

return count; //return count value

}

# int menu()

{

int x; //declare variable

char proceed; //declare variable

cout << "\n--------------Welcome to the Galaxy of Songs--------------\n"

<< "==========================================================\n"

<< "\t\t\t Menu\n"

<< "==========================================================\n"

<< "\t\t1) Add new song\n"

<< "\t\t2) Remove song\n"

<< "\t\t3) Update or edit song\n"

<< "\t\t4) List of songs\n"

<< "\t\t5) Search a song\n"

<< "\t\t6) Create random playlist\n"

<< "\t\t7) Exit program\n"

<< "----------------------------------------------------------\n"; //show menu

do

{

if (cin.fail()) //prevent user from inputing value other than integer

{

cin.clear();

while (getchar() != '\n');

}

cout << "What do you wish to do? (Please enter the number on the menu): ";

cin >> x;

cout << "\n";

if (x < 1 || x > 7)

cout << "INVALID INPUT ! PLEASE ENTER 1 TO 7 ONLY!\n\n";

} while (x < 1 || x > 7); //data validation, 1 - 7 only

do { //data validation to ask user to proceed chosen menu, or back to menu

cout << "\nDirecting you to the menu of choice...\nPress Y to proceed or N to return menu: ";

cin >> proceed;

if (tolower(proceed) != 'y' && tolower(proceed) != 'n')

cout << "ERROR: PLEASE ENTER Y OR N ONLY" << endl;

} while (tolower(proceed) != 'y' && tolower(proceed) != 'n');

if (tolower(proceed) == 'y') //if user input y(proceed), then return x

return x;

else if (tolower(proceed) == 'n') //if user input x(return menu), call menu()

menu();

}

# void addSong((struct COLLECTION newSong[], int count)

# {

# DATE tempdate;

# cout << endl;

# cout << "-----------------------Add New Song-----------------------\n"; //show header of Add New Song

# do

# {

# cin.ignore(); //use to ignore input buffer

# cout << "Please enter the song's title: ";

# getline(cin, newSong[count].title); //input statement for song title

# if (newSong[count].title.empty()) //if user input is empty will prompt out error statement

# {

# cout << "ERROR: PLEASE INPUT SONG'S TITLE! PRESS ENTER TO CONTINUE..." << endl; //error message

# }

# } while (newSong[count].title.empty()); //(data validation) when user input song title is empty, keep on looping until user input valid data

# do

# {

# cout << "Please enter the song's album: ";

# getline(cin, newSong[count].album); //input statement

# if (newSong[count].album.empty())

# {

# cout << "ERROR: PLEASE INPUT SONG'S ALBUM!" << endl; //error message

# }

# } while (newSong[count].album.empty()); //data validation

# do

# {

# cout << "Please enter the song's artist: ";

# getline(cin, newSong[count].artist); //input statement

# if (newSong[count].artist.empty())

# {

# cout << "ERROR: PLEASE INPUT SONG'S ARTIST!" << endl; //error message

# }

# } while (newSong[count].artist.empty()); //data validation

# do

# {

# cout << "Please enter the song's composer: ";

# getline(cin, newSong[count].composer); //input statement

# if (newSong[count].composer.empty())

# {

# cout << "ERROR: PLEASE INPUT SONG'S COMPOSER!" << endl; //error message

# }

# } while (newSong[count].composer.empty()); //data validation

# do //data validation

# {

# cout << "Please enter the song's genre: ";

# getline(cin, newSong[count].genre); //input statement

# if (newSong[count].genre.empty())

# {

# cout << "ERROR: PLEASE INPUT SONG'S GENRE!" << endl; //error message

# }

# } while (newSong[count].genre.empty());

# do {

# do

# {

# cout << "Please enter the song's release day: ";

# cin >> tempdate.day; //input statement

# if (cin.fail()) //prevent user from input value other then integer

# {

# cin.clear();

# while (getchar() != '\n');

# }

# if (tempdate.day < 1 || tempdate.day > 31)

# {

# cout << "ERROR: PLEASE ENTER VALID DAY!" << endl; //error message

# }

# } while (tempdate.day < 1 || tempdate.day > 31);

# 

# do {

# cout << "Please enter the song's release month: ";

# cin >> tempdate.month; //input statement

# if (cin.fail()) //prevent user from input value other then integer

# {

# cin.clear();

# while (getchar() != '\n');

# }

# if (tempdate.month < 1 || tempdate.month > 12)

# {

# cout << "ERROR: PLEASE ENTER VALID MONTH!" << endl; //error message

# }

# else if ((tempdate.month == 4 || tempdate.month == 6 || tempdate.month == 9 || tempdate.month == 11) && tempdate.day == 31) //date validation

# {

# cout << "ERROR: INVALID DAY! PLEASE ENTER AGAIN!" << endl; //error message

# }

# } while (tempdate.month < 1 || tempdate.month > 12);

# } while ((tempdate.month == 4 || tempdate.month == 6 || tempdate.month == 9 || tempdate.month == 11) && (tempdate.day == 31)); //date validation

# 

# do {

# do

# {

# cout << "Please enter the song's release year: ";

# cin >> tempdate.year; //input statement

# if (cin.fail()) //prevent user from input value other then integer

# {

# cin.clear();

# while (getchar() != '\n');

# }

# if (tempdate.year < 1900 || tempdate.year > 2050)

# {

# cout << "ERROR: PLEASE ENTER A VALID YEAR!" << endl; //error message

# }

# else if ((tempdate.month == 2) && (tempdate.year % 4 == 0) && tempdate.day > 29) {

# cout << "ERROR: INVALID YEAR! PLEASE ENTER AGAIN!" << endl; //error message

# }

# else if ((tempdate.month == 2) && (tempdate.year % 4 != 0) && tempdate.day > 28) {

# cout << "ERROR: INVALID YEAR! PLEASE ENTER AGAIN!" << endl; //error message

# }

# } while (tempdate.year < 1900 || tempdate.year > 2050);

# } while ((tempdate.month == 2) && (tempdate.year % 4 == 0) && (tempdate.day > 29) || ((tempdate.month == 2) && (tempdate.year % 4 != 0) && tempdate.day > 28)); //date validation

# 

# do //data validation

# {

# cin.ignore();

# cout << "Please enter the song's directory path (with .mp3): ";

# getline(cin, newSong[count].directory); //input statement

# if (newSong[count].directory.empty())

# {

# cout << "ERROR: PLEASE INPUT THE DIRECTORY PATH!" << endl; //error message

# }

# } while (newSong[count].directory.empty());

# newSong[count].releaseDate = to\_string(tempdate.day) + "/" + to\_string(tempdate.month) + "/" + to\_string(tempdate.year);

# }

# 

# void deleteSong(struct COLLECTION musicList[], int count)

{

string removeSong; //declare removeSong to store user input

bool found = false; //initialize boolean type found to false

cout << "Enter a song title that you want to delete:";

cin.ignore(); //ignore input buffer

getline(cin, removeSong); //get user input

string lowerRemove = removeSong; //declare lowerRemove

//transform user input to lowercase

transform(lowerRemove.begin(), lowerRemove.end(), lowerRemove.begin(), ::tolower);

for (int i = 0; i < count ; i++) //loop through struct

{

string lowerTitle = musicList[i].title; //declare lowerTitle as musicList[i].title

transform(lowerTitle.begin(), lowerTitle.end(), lowerTitle.begin(), ::tolower);

if (lowerTitle == lowerRemove) //if lowerTitle equal to user input

{

for (int j = i; j < count; j++)

{

musicList[j] = musicList[j + 1];

}

found = true; //declare found to true if song is found

}

}

if (!found) //if found is false

{

cout << "This song is not in the list!" << endl; //error message

}

else

{

cout << "Successfully deleted!" << endl;

}

}

# void updateSong();

# void listSong(struct COLLECTION listSong[], int count)

{

//display "header"

cout << right << setw(5) << "NO. " << left << setw(35) << "SONG TITLE" << "| " << left << setw(25) << "ALBUM" << "| " << left << setw(20) << "ARTIST" << "| " << left << setw(20) <<

"COMPOSER" << "| " << left << setw(15) << "GENRE" << "|" << right << setw(13) << "RELEASE DATE" << " | " << left << setw(70) << "DIRECTORY PATH" << endl;

cout << "-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

for (int i = 0; i < count; i++) { //for i less than count, display the array of structure

cout << right << setw(3) << i + 1 << ". " << left << setw(35) << listSong[i].title << "| ";

cout << left << setw(25) << listSong[i].album << "| ";

cout << left << setw(20) << listSong[i].artist << "| ";

cout << left << setw(20) << listSong[i].composer << "| ";

cout << left << setw(15) << listSong[i].genre;

cout << "|" << right << setw(13) << listSong[i].releaseDate << " | ";

cout << left << setw(70) << listSong[i].directory << endl;

}

cout << "\nYOU HAVE A TOTAL OF " << count << " SONGS IN THIS PROGRAM :)" << endl; //show total songs in collection

}

# void searchSong(struct COLLECTION musicList[], int count)

{

string searchSong; //declare searchSong to store user input

bool ans = false; //initialize boolean type ans to false

cout << "Enter song title / artist / album to search for a song:";

cin.ignore(); //use to ignore input buffer

getline(cin, searchSong); //get user input

string lowerSearch = searchSong;

//transform user input to lowercase

transform(lowerSearch.begin(), lowerSearch.end(), lowerSearch.begin(), ::tolower);

//print header

cout << right << setw(5) << "NO. " << left << setw(35) << "SONG TITLE" << "| " << left << setw(25) << "ALBUM" << "| " << left << setw(20) << "ARTIST" << "| " << left << setw(20) <<

"COMPOSER" << "| " << left << setw(15) << "GENRE" << "|" << right << setw(13) << "RELEASE DATE" << " | " << left << setw(70) << "DIRECTORY PATH" << endl;

cout << "----------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

for (int i = 0; i < count ; i++)

{

string lowerTitle = musicList[i].title;

string lowerAlbum = musicList[i].album;

string lowerArtist = musicList[i].artist;

transform(lowerTitle.begin(), lowerTitle.end(), lowerTitle.begin(), ::tolower);

transform(lowerAlbum.begin(), lowerAlbum.end(), lowerAlbum.begin(), ::tolower);

transform(lowerArtist.begin(), lowerArtist.end(), lowerArtist.begin(), ::tolower);

//if user input partially or fully matched with lowerTitle or lowerAlbum or lowerArtist

if (lowerTitle.find(lowerSearch) != string::npos || lowerAlbum.find(lowerSearch) != string::npos || lowerArtist.find(lowerSearch) != string::npos)

{

ans = true; //declare ans to true is song found

cout << right << setw(3) << i + 1 << ". " << left << setw(35) << musicList[i].title << "| ";

cout << left << setw(25) << musicList[i].album << "| ";

cout << left << setw(20) << musicList[i].artist << "| ";

cout << left << setw(20) << musicList[i].composer << "| ";

cout << left << setw(15) << musicList[i].genre;

cout << "|" << right << setw(13) << musicList[i].releaseDate << " | ";

cout << left << setw(70) << musicList[i].directory << endl;

}

}

if (ans != true) //if song does not found assign false to ans

{

cout << "NO SONG FOUND!!!" << endl; //error message

}

}

# 

# void genPlaylist(struct COLLECTION songList[], int count)

{

vector<string> artistList{}; //To create vector of artist and genre

vector <string> genreList{};

for (int i = 0; i < count - 1; i++)

{

artistList.push\_back(musicList[i].artist); //To initialize vector artistList with every artist in songList structure

genreList.push\_back(musicList[i].genre); //To initialize vector genreList with every genre in songList structure

}

sort(artistList.begin(), artistList.end()); //To sort vector list from a-z ascendingly

artistList.erase(unique(artistList.begin(), artistList.end()), artistList.end()); //To erase duplicates of strings inside vector list

sort(genreList.begin(), genreList.end());

genreList.erase(unique(genreList.begin(), genreList.end()), genreList.end());

int sortChoice;

string artistChoice, genreChoice;

int artistCount = 0, genreCount = 0;

vector<int> countResult{}; //Vector list with integer type to be able to shuffle the songs

srand(unsigned(time(NULL))); //Seed for random generator (time)

cout << "\nDo you want to sort by Artist or Genre?\n";

cout << "1. Artist\n2. Genre\n(Enter 1 or 2)\n";

cin >> sortChoice;

while (sortChoice < 1 || sortChoice > 2) { //Data validation: Only integers 1 and 2 are accepted

if (!cin) {

cin.clear();

cin.ignore(10000, '\n');

cout << "You didn't enter an integer! Please try again: ";

cin >> sortChoice;

}

else {

cout << "Please enter 1 or 2: ";

cin.ignore(10000, '\n');

cin >> sortChoice;

}

}

switch (sortChoice)

{

case 1:

cout << "\nBelow are the list of artists:\n"; //Displays the list of unique artists in songList structure

for (string x : artistList) {

cout << x << endl;

}

cout << "\nWhich artist do you want to sort by?\n";

cin.ignore();

getline(cin, artistChoice);

do { //Data validation

transform(artistChoice.begin(), artistChoice.end(), artistChoice.begin(), ::tolower); //To turn user-input to all lowercase for search function

for (int k = 0; k < count - 1; k++) {

transform(musicList[k].artist.begin(), musicList[k].artist.end(), musicList[k].artist.begin(), ::tolower); //To turn vector list strings to lowercase to match user-input

if (musicList[k].artist.find(artistChoice) != string::npos) {

countResult.push\_back(k);

}

}

if (countResult.size() < 1) {

cout << "Your input didn't match any results! Please enter a new artist: ";

getline(cin, artistChoice);

}

} while (countResult.size() < 1);

random\_shuffle(countResult.begin(), countResult.end()); //Shuffles the integers in the vector list (This is where the song shuffle begins)

if (countResult.size() > 10) { //If theres more than 10 songs in the vector

countResult.resize(10); //Resizes the vector list to 10 integers only (so 10 songs are generated, not more)

cout << "\nYour generated playlist of 10 songs sorted by artist: \"" << artistChoice << "\" contains:\n\n";

}

else if (countResult.size() < 10) {

cout << "\nYour specified artist doesn't have enough songs! Here is a generated playlist of " << countResult.size() << " songs: \n\n";

}

cout << left << setw(35) << "SONG TITLE" << "| " << left << setw(25) << "ALBUM" << "| " << left << setw(20) << "ARTIST" << "| " << left << setw(20) << "COMPOSER" << "| " << left <<

setw(15) << "GENRE" << "|" << right << setw(13) << "RELEASE DATE" << " | " << left << setw(70) << "DIRECTORY PATH" << endl;

cout << "-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

for (int display : countResult) { //Displays the newly generated playlist

for (int z = 0; z < musicList[display].artist.length(); z++) { //To recapitalize the string that has been lower-cased

if (z == 0) {

musicList[display].artist[z] = toupper(musicList[display].artist[z]);

}

else if (musicList[display].artist[z - 1] == ' ') {

musicList[display].artist[z] = toupper(musicList[display].artist[z]);

}

}

cout << right << setw(3) << left << setw(35) << musicList[display].title << "| ";

cout << left << setw(25) << musicList[display].album << "| ";

cout << left << setw(20) << musicList[display].artist << "| ";

cout << left << setw(20) << musicList[display].composer << "| ";

cout << left << setw(15) << musicList[display].genre;

cout << "|" << right << setw(13) << musicList[display].releaseDate << " | ";

cout << left << setw(70) << musicList[display].directory << endl;

}

break;

case 2:

cout << "\nBelow are the list of genre:\n"; //Displays the list of unique genres in songList structure

for (string y : genreList) {

cout << y << endl;

}

cout << "\nWhich genre do you want to sort by?\n";

cin.ignore();

getline(cin, genreChoice);

do { //Data validation

transform(genreChoice.begin(), genreChoice.end(), genreChoice.begin(), ::tolower); //To turn user-input to all lowercase for search function

for (int l = 0; l < count - 1; l++) {

transform(musicList[l].genre.begin(), musicList[l].genre.end(), musicList[l].genre.begin(), ::tolower); //To turn vector list strings to lowercase to match user-input

if (musicList[l].genre.find(genreChoice) != string::npos) {

countResult.push\_back(l);

}

}

if (countResult.size() < 1) {

cout << "Your input didn't match any results! Please enter a new genre: ";

getline(cin, genreChoice);

}

} while (countResult.size() < 1);

random\_shuffle(countResult.begin(), countResult.end()); //Shuffles the integers in the vector list (This is where the song shuffle begins)

if (countResult.size() > 10) { //If theres more than 10 songs in the vector

countResult.resize(10); //Resizes the vector list to 10 integers only (so 10 songs are generated, not more)

cout << "\nYour generated playlist of 10 songs sorted by genre: \"" << genreChoice << "\" contains:\n\n";

}

else if (countResult.size() < 10) {

cout << "\nYour specified genre doesn't have enough songs! Here is a generated playlist of " << countResult.size() << " songs: \n\n";

}

cout << left << setw(35) << "SONG TITLE" << "| " << left << setw(25) << "ALBUM" << "| " << left << setw(20) << "ARTIST" << "| " << left << setw(20) << "COMPOSER" << "| " << left <<

setw(15) << "GENRE" << "|" << right << setw(13) << "RELEASE DATE" << " | " << left << setw(70) << "DIRECTORY PATH" << endl;

cout << "-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------" << endl;

for (int display : countResult) { //Displays the newly generated playlist

for (int z = 0; z < musicList[display].genre.length(); z++) { //To recapitalize the string that has been lower-cased

if (z == 0) {

musicList[display].genre[z] = toupper(musicList[display].genre[z]);

}

else if (musicList[display].genre[z - 1] == ' ') {

musicList[display].genre[z] = toupper(musicList[display].genre[z]);

}

}

cout << right << setw(3) << left << setw(35) << musicList[display].title << "| ";

cout << left << setw(25) << musicList[display].album << "| ";

cout << left << setw(20) << musicList[display].artist << "| ";

cout << left << setw(20) << musicList[display].composer << "| ";

cout << left << setw(15) << musicList[display].genre;

cout << "|" << right << setw(13) << musicList[display].releaseDate << " | ";

cout << left << setw(70) << musicList[display].directory << endl;

}

break;

}

}

# void exitApp(struct COLLECTION musicList[], int count)

{

ofstream write\_file("songlist.txt"); //declare output file stream

if (!write\_file) //if file not found

{ //error message

cout << "ERROR: FILE CANNOT BE OPENED!" << endl;

}

else

{

write\_file << "Song Title,Album,Artist,Composer,Genre,Release Date,Directory Path"; //diaply "header" in file

for (int i = 0; i < count; i++)

{

write\_file << "\n" << musicList[i].title << "," << musicList[i].album << "," << musicList[i].artist << "," << musicList[i].composer << "," << musicList[i].genre << "," << musicList[i].releaseDate << "," << musicList[i].directory;

} //write data in array of structure back to text file

}

write\_file.close(); //close file

cout << "Saving changes made...\n.\n.\n.\nSaved!\n" << endl;

cout << "Are you sure to exit Galaxy of Song? "; //ask user for exit conformation

}