```
//Ben Scherer
// 8/3/2017
// Final Project
// Uses delimited text file as movie "Database".  ALlows search,update,delete,create functions.
// *** Leverages struct instead of paralelle arrays


//Headers to include
#include <iostream> //cout
#include <iomanip> // used to manipulate cout
#include <string> //needed for string variable
#include <math.h> //used for basic arithmatic
#include <limits> //user for numeric_limits
#include <vector> //needed to use vectors
#include <fstream> //file handling
#include <sstream> //used for string bufffer
#include <cstddef>

using namespace std;

//data structure for each individual movie
struct Movie {
    string title; //name of movie
    string studio; //name of studio
    string contentRating;
    int year; //year of release
    float rating; //10 star rating system
    string genre; //genre of movie
    string releaseDate; //date of release
    string writers; //writers of movie
    string runtime; //runtime
    string directors; //directors
    string actors; //actors



};

//Functions
void displayWelcome();
void movieDetails(vector<Movie> &movies, int movieIndex); //display details about movie
int getIntInput(string questionToAsk, string errorMsg); //validates int input and int range
int getIntInput(string questionToAsk, string errorMsg, int lowRange, int highRange); //validates int input and int range
string getInput(string questionToAsk, string errorMsg); //validates string input
Movie parseData(string movieString); //parses data from comma delimted string and returns Movie object
vector<Movie> loadDB(string strFileName); //loads data from flat file to vector
Movie addMovie(); //add movie to database
```

```cpp
vector<int> searchTitle(vector<Movie> movies); //search for movies by title
vector<int> searchGenre(vector<Movie> movies);
vector<int> searchActor(vector<Movie> movies);
vector<int> searchYear(vector<Movie> movies);
vector<int> searchDirector(vector<Movie> movies);
void searchMovies(vector<Movie> &searchVector);
char getSentinel(string message,string errMessage); //Input validation for sentinel.  simple y/n
Movie updateMovie(Movie mvObj); //update movie
void displayMovie(Movie mvObj); //display movie
float getFloatInput(string questionToAsk, string errorMsg, int lowRange, int highRange);
void updateDB(vector<Movie> movies,string file); //write chagnes to file



int main() {
    const string dbFile = "movies.csv";
    vector<Movie> movies; //contents of DB will be loaded into vector for runtime
    //cout << "Welcome to the CIS111 offline movie database\n";
    movies = loadDB(dbFile);

    int menuChoice;
    displayWelcome();
    do {

        menuChoice = getIntInput("", "Error: Enter valid number");
        switch (menuChoice) {
        case 1: searchMovies(movies); break;
        case 2: movies.push_back(addMovie());

        }
        displayWelcome();
    } while (menuChoice == 1 || menuChoice == 2);

    cout << "Writing changes to database file\n";
    updateDB(movies, dbFile);

    system("pause");
}

//write changes to db file
void updateDB(vector<Movie> movies,string file) {
    ofstream outFile(file);
    for (Movie mvObj : movies) {
        //title,studio,content,year,rating,genre,releasedate,writers,runtime,directors,actors
        outFile << mvObj.title << "|" << mvObj.studio << "|" << mvObj.contentRating << "|" << mvObj.year << "|" << mvObj.rating << "|" << mvObj.genre << "|" << mvObj.releaseDate << "|" << mvObj.writers << "|" << mvObj.runtime << "|" << mvObj.directors << "|" << mvObj.actors
```

```cpp
                << endl;
    }
    outFile.close();
}



//Add movie to database
Movie addMovie() {
    Movie mvObj;
    mvObj.title = getInput("Enter new title: ", "ERror: enter valid string");
    mvObj.year = getIntInput("Enter new Year: ", "Error: Enter valid year");
    mvObj.releaseDate = getInput("Enter new release date: ", "Error: Enter valid 
      date");
    mvObj.contentRating = getInput("Enter new content rating: ", "Error: Enter 
      valid rating");
    mvObj.genre = getInput("Enter new genre:  ", "Error: Enter valid string");
    mvObj.directors = getInput("Enter new directors: ", "Error: Enter valid 
      string");
    mvObj.writers = getInput("Enter new writers: ", "Error: Enter valid string");
    mvObj.actors = getInput("Enter new actors: ", "Error: Enter valid string");
    mvObj.rating = getFloatInput("Enter new rating(0-10): ", "Error: Enter valid 
      number", 0.0, 10.0);
    return mvObj;

}

void searchMovies(vector<Movie> &searchVector) {

    int menuChoice;
    vector<int> foundMovies;
    cout << "Please select search type.\n";
    cout << "1. By Title\n"
        << "2. By Actor\n"
        << "3. By Year\n"
        << "4. By Genre\n"
        << "Enter a number between 1 -5.  Any other number will  \n";

        menuChoice = getIntInput("", "Error: Enter valid number");

        if (menuChoice < 1 || menuChoice > 5) return;

    switch (menuChoice) {
    case 1: foundMovies = searchTitle(searchVector); break;
    case 2: foundMovies = searchActor(searchVector); break;
    case 3: foundMovies = searchYear(searchVector); break;
    case 4: foundMovies = searchGenre(searchVector); break;

    }
    int count = 0;
    if (!foundMovies.empty()) {
        cout << "Matching Movies found:\n";
        for (int i : foundMovies) {

            cout <<count << ": " << searchVector[foundMovies[count]].title << endl;
            count++;
        }
        menuChoice = getIntInput("Enter number of movie to view details", "Error: Enter valid number", 0, count);
        movieDetails(searchVector,foundMovies[menuChoice]);

    }
    else {
        cout << "No movies matched the search string\n";
    }
}

vector<int> searchTitle(vector<Movie> movies) {
    string searchString = getInput("Enter title of movie to search for: ", "Error: Enter valid string\n");
    vector<int> movieIndex;
    int count = 0;
    for (Movie mvObj : movies) {
    //  cout << mvObj.title.find(searchString) << mvObj.title << endl;
        if (mvObj.title.find(searchString) != string::npos) {
            movieIndex.push_back(count);
        }
        count++;
    }

    return movieIndex;
}

//search by actor
vector<int> searchActor(vector<Movie>movies) {
    string searchString = getInput("Enter title of movie to search for: ", "Error: Enter valid string\n");
    vector<int> movieIndex;
    int count = 0;
    for (Movie mvObj : movies) {
        //  cout << mvObj.title.find(searchString) << mvObj.title << endl;
        if (mvObj.actors.find(searchString) != string::npos) {
            movieIndex.push_back(count);
        }
        count++;
    }

    return movieIndex;
}

//search by year
vector<int> searchYear(vector<Movie>movies) {
    int searchString = getIntInput("Enter new Year", "Error: Enter valid year");
    vector<int> movieIndex;
    int count = 0;
    for (Movie mvObj : movies) {
        //  cout << mvObj.title.find(searchString) << mvObj.title << endl;
        if (mvObj.year == searchString) {
            movieIndex.push_back(count);
        }
        count++;
    }

    return movieIndex;
}

//search by genre
vector<int> searchGenre(vector<Movie>movies) {
    string searchString = getInput("Enter title of movie to search for: ",
      "Error: Enter valid string\n");
    vector<int> movieIndex;
    int count = 0;
    for (Movie mvObj : movies) {
        //  cout << mvObj.title.find(searchString) << mvObj.title << endl;
        if (mvObj.genre.find(searchString) != string::npos) {
            movieIndex.push_back(count);
        }
        count++;
    }

    return movieIndex;
}

vector<Movie> loadDB(string strFileName) {
    /* Reads db file in and loads into vector
    http://www.fluentcpp.com/2017/04/21/how-to-split-a-string-in-c/
    */
    vector<Movie> movies; //holds movie data
    ifstream dbFile(strFileName); //Open DB File


    if (dbFile.fail()) {
        cout << "ERROR: Unable to open database file: " << strFileName << endl;
        exit(1);
    }
    cout << "Loading database.....\n";
    string strHold; //placeholder for getline
    while (getline(dbFile, strHold)) {
        //cout << strHold << endl;
        movies.push_back(parseData(strHold));

    }
    dbFile.close();
    return movies;

```

```cpp
}

//Parsed movie data from comma seperated string
////http://www.fluentcpp.com/2017/04/21/how-to-split-a-string-in-c/
Movie parseData(string movieString) {
    istringstream strStream(movieString); //string stream var, needed to split
    string tempStr; //temporary holder for splitting string
    vector<string> parsedItems; //vector to hold the parsed data
    while (getline(strStream, tempStr, '|')) {
        parsedItems.push_back(tempStr);
    }

    //populate Movie struct with data
    Movie movieObj;
    //cout << parsedItems[0] << endl;
    movieObj.title = parsedItems[0];
    movieObj.studio = parsedItems[1];
    movieObj.contentRating = parsedItems[2];
    try {
        movieObj.year = stoi(parsedItems[3]);
    }
    catch (exception e) {
        movieObj.year = 0000;
    }
    try {
        movieObj.rating = stof(parsedItems[4]);
    }
    catch (exception e) {
        movieObj.rating = 0.0;
    }

    movieObj.genre = parsedItems[5];
    movieObj.releaseDate = parsedItems[6];
    movieObj.writers = parsedItems[7];
    movieObj.runtime = parsedItems[8];
    movieObj.directors = parsedItems[9];
    movieObj.actors = parsedItems[10];

    return movieObj;



}

//Displays initial welcome screen
void displayWelcome() {
    cout << "---------------------------------------------------------------------------\n";
    cout << "Welcome to the Offline Movie Database.   Please choose from an option below\n";
    cout <<
```

```
        "----------------------------------------------------------------------------\n";
    cout << "1. Search for a movie\n"
        << "2. Add new movie\n"
        << "Any other number to quit\n";
}

void movieDetails(vector<Movie> &movies,int movieIndex) {
    displayMovie(movies[movieIndex]);

    int menuChoice = getIntInput("Enter 1 to update movie.  Enter 2 to delete movie.  Enter any other number to return to menu\n", "Error: Enter valid number");
    switch (menuChoice) {
    case 1: {
        movies[movieIndex] = updateMovie(movies[movieIndex]);
        break;
        }
    case 2: {
        cout << "Removing " << movies[movieIndex].title << " from data base\n";
        movies.erase(movies.begin() + movieIndex);

        }
    }


}

//display movie details
void displayMovie(Movie mvObj) {
    cout << "---------------------------------------------------------------\n";
    cout << "\tMovie Details\n";
    cout << "---------------------------------------------------------------\n";
    cout << left << setw(20) << "0 - Title: " << mvObj.title << endl;
    cout << left << setw(20) << "1 - Year: " << mvObj.year << endl;
    cout << left << setw(20) << "2 - Release: " << mvObj.releaseDate << endl;
    cout << left << setw(20) << "3 - Content Rating: " << mvObj.contentRating << endl;
    cout << left << setw(20) << "4 - Genre: " << mvObj.genre << endl;
    cout << left << setw(20) << "5 - Director: " << mvObj.directors << endl;
    cout << left << setw(20) << "6 - Writers: " << mvObj.writers << endl;
    cout << left << setw(20) << "7 - Actors: " << mvObj.actors << endl;
    cout << left << setw(20) << "8 - Rating: " << mvObj.rating << endl;
}

//Update fields of movie
Movie updateMovie(Movie mvObj) {
    int menuChoice;
    do {


        menuChoice = getIntInput("Enter number of field to modify: \n", "Error:
```

```cpp
          Enter valid number", 0, 8);
        switch (menuChoice) {
            case 0: mvObj.title = getInput("Enter new title: ", "ERror: enter valid string"); break;
            case 1: mvObj.year = getIntInput("Enter new Year: ", "Error: Enter valid year"); break;
            case 2: mvObj.releaseDate = getInput("Enter new release date: ", "Error: Enter valid date"); break;
            case 3: mvObj.contentRating = getInput("Enter new content rating: ", "Error: Enter valid rating"); break;
            case 4: mvObj.genre = getInput("Enter new genre: ", "Error: Enter valid string"); break;
            case 5: mvObj.directors = getInput("Enter new directors: ", "Error: Enter valid string"); break;
            case 6: mvObj.writers = getInput("Enter new writers: ", "Error: Enter valid string"); break;
            case 7: mvObj.actors = getInput("Enter new actors: ", "Error: Enter valid string"); break;
            case 8: mvObj.rating = getFloatInput("Enter new rating(0-10): ", "Error: Enter valid number",0,10); break;

        }
        displayMovie(mvObj);

    } while (getSentinel("Enter 'y' to make further updates.  Enter 'n' to exit: ","Error: Enter 'y'") == 'y');
    return mvObj;
}

//Validates input based on a range.  returns int
int getIntInput(string questionToAsk, string errorMsg) {
    int usrInput;
    cout << questionToAsk;
    while (!(cin >> usrInput)  ) { //Loop until integer in the specified range is entered
        cout << errorMsg << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
    cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    return int(usrInput);
}

//Validates input based on a range.  returns float
float getFloatInput(string questionToAsk, string errorMsg, int lowRange, int highRange) {
    float usrInput;
    cout << questionToAsk;
    while (!(cin >> usrInput) || usrInput < lowRange || usrInput > highRange) { //Loop until integer in the specified range is entered
        cout << errorMsg << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
    cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    return usrInput;
}


//Validates input based on a range.  returns int
int getIntInput(string questionToAsk, string errorMsg,int lowRange,int highRange) {
    int usrInput;
    cout << questionToAsk;
    while (!(cin >> usrInput) || usrInput < lowRange || usrInput > highRange) { //Loop until integer in the specified range is entered
        cout << errorMsg << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }
    cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    return int(usrInput);
}

//Validates input, returns string
string getInput(string questionToAsk, string errorMsg) {
    string usrInput;
    cout << questionToAsk;
    cin.sync();

    getline(cin, usrInput);

    return usrInput;
}




//validates sentinel input, then returns char value
char getSentinel(string question,string errMessage) {
    char  varToReturn;
    bool isValidInput = false;

    // loop until a valid y or n char is entered
    do {
        cout << question;
        if (!(cin >> varToReturn) || (tolower(varToReturn) != 'y' && tolower(varToReturn) != 'n')) {
            cout << errMessage;

            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        else {
            isValidInput = true;
        }
    } while (!isValidInput);
    cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    return tolower(varToReturn);
}
```