Developer Documentation

The **Songs** program is designed to manage a database of songs, providing users with the ability to store, modify, and retrieve detailed information about their music collection. The program offers a range of functionalities that allow for efficient management and querying of song data.

Functional Components

User Interface Module

Main File: main.c

- **Purpose**: Facilitates interaction between the user and the program through a menu-driven interface.
- **Features**:
  - Presents a clear and intuitive main menu with options corresponding to each functionality.
  - Handles user input to navigate between different operations.
  - Provides prompts and feedback to guide the user through each process.

Data Management Module

Main File: song_database.c **with** song_database.h.

- **Purpose**: Manages the storage, addition, deletion, and modification of song entries within the database.

Features:

1. **Load Existing Database**

   `void loadDatabase(SongDatabase *db, const char *filename):`

   - **Functionality**: Allows users to load song entries from an existing file into the program. If no file exists, it initializes a new database.
   - **Input**: db (pointer to database), filename (file to load).
   - **Output**: Populates the database or leaves it empty.

2. **Add New Song Entry**

   `void addSong(SongDatabase *db):`

   - **Functionality**: Collects detailed information about a new song from the user and adds it to the database.
   - **Benefit**: Keeps the database up-to-date with the latest additions to the user's music collection.
   - **Input:** User-provided details via scanf.
   - **Output:** Song added to the database.

3. **Delete Song Entry**

   `void deleteSong(SongDatabase *db):`

   - **Functionality**: Removes a song from the database based on criteria such as the song's title.
   - **Benefit**: Helps maintain an accurate and organized database by removing outdated or unwanted entries.
   - **Input:** db (database), user input for title.
   - **Output:** Removes the song if found, displays a message otherwise.

4. **Edit Song Entry**

   `void editSong(SongDatabase *db):`

   - **Functionality**: Provides the ability to modify details of an existing song in the database.
   - **Benefit**: Ensures that song information remains current and correct.
   - **Input:** db (database), user input for the title and fields to modify.
   - **Output:** Updates the song in the database.

5. **Save Database to File**

   `void saveDatabase(SongDatabase *db, const char *filename);`

   - **Functionality**: Saves the current state of the database to a file specified by the user.
   - **Benefit**: Preserves the user's data for future sessions, preventing loss of information.

  - **Input**: db (database with a filename).
  - **Output:** Writes the database to the file.

Query Module

Main File: song_query.c with song_query.h.

- **Purpose**: Provides functionalities to search and display song entries based on specific criteria.
- **Features**:
  - **Display Songs by Artist**: Lists all songs performed by a user-specified artist.
  - **Display Songs by Album**: Shows all details of songs from a particular album by a specified artist.
  - **List Songs by Year**: Retrieves all songs released in a user-selected year.
  - **List Songs by Genre**: Displays all songs belonging to a user-selected genre.

Query Functions in `song_query.c`

1. **`void displaySongsByArtist(const SongDatabase *db):`**
   - Displays all songs matching the artist name.
2. **`void displaySongsByAlbum(const SongDatabase *db):`**
   - Displays all songs from a specific album.
3. **`void displaySongsByYear(const SongDatabase *db):`**
   - Displays all songs released in a specific year.
4. **`void displaySongsByGenre(const SongDatabase *db):`**
   - Displays all songs of a specific genre.

Key Functionalities

6. **Display Songs by Artist**
   - **Functionality**: Lists all songs in the database performed by a specific artist entered by the user.

  - **Benefit**: Allows users to quickly find and review all songs by their favorite artists.
  -

7. **Display Songs by Album**
   - **Functionality**: Shows detailed information for all songs on a specific album by a particular artist.
   - **Benefit**: Enables users to explore the contents of specific albums in their collection.
8. **List Songs by Release Year**
   - **Functionality**: Retrieves and displays all songs released in a user-selected year.
   - **Benefit**: Helps users discover music from a particular time period.
9. **List Songs by Genre**
   - **Functionality**: Displays all songs within the database that belong to a specific genre chosen by the user.
   - **Benefit**: Assists users in exploring and organizing their music based on genre preferences.

Data Handling

Data Structures

1. Song Structure

Defined in song_database.h:

```
typedef struct {
    char title[50];
    char artist[50];
    char album[50];
    int release_year;
    char genre[20];
    int length_minutes;
    int length_seconds;
} Song;
```

- Stores all relevant details about a song.

2. SongDatabase Structure

Defined in song_database.h:

```
typedef struct {
    Song *songs;
    int size;
    int capacity;
    char filename[100];
} SongDatabase;
```

- **Dynamic Array**: Stores an expandable collection of songs.
- **Capacity Management**: Uses realloc to grow the array as needed.
- **Filename**: Tracks the associated file for saving and loading.

Song Information Stored:

- Title of the song
- Name of the performer (singer or group)
- Title of the album it was released on
- Year of release
- Genre of the music (e.g., rock, pop, hip-hop, jazz, classical)
- Length of the song (in minutes and seconds)

Database Characteristics:

- **Dynamic Storage**: The database can handle an arbitrary number of song entries, accommodating the user's growing music collection.
- **Persistent Storage**: Data can be saved to and loaded from files, ensuring that the user's collection is preserved between sessions.
- **Efficient Management**: Provides quick access to song data and supports efficient updating of the database through adding, editing, and deleting entries.

Requirements to Build

Language: C

Compiler: GCC (or any C standard-compliant compiler)

Standard libraries: <stdio.h>, <stdlib.h>, <string.h>.

Custom headers: song_database.h, song_query.h.

Design Considerations

1. **Dynamic Memory Management**:
    - Avoids fixed limits on the number of songs by dynamically resizing the database.
    - Uses efficient reallocation techniques.
2. **Separation of Concerns**:
    - Keeps interface, core logic, and query functionality in distinct modules.
3. **Error Handling**:
    - Handles file errors (missing or unreadable files).
    - Ensures safe memory operations with checks on malloc and realloc.
4. **Code Reusability**:
    - Helper functions like printSongInfo prevent redundant code.