Developer Documentation

Daud Iqbal
X32SQC

SONGS

This is a database program that is built by using extensive features of C language. This program can efficiently make a reasonable collection of songs and store it into memory and can access at any time with just a call. The main objectives for the program were defined in the specifications. Let's have a look into the formation of this code, the ideas and thinking process behind it and how they take place in the compiler.

The program is capable of performing the following tasks:

- Create a new database
- Store data entries in the form of songs
- Load the existing data base
- Search for a particular song in database with any one of the given parameters
- Inserting new songs into the database
- Print out the whole database
- The program is capable of recommending songs of a given genre
- The program can find out the most popular songs present in the database
- The program also consists of a timer function that controls transitions on the console screen for smoother output.

The program is highly based upon the concepts of Data structures, File handling, Dynamic Memory Allocation and String manipulation. To make the code easier the whole program has been divided into segments and modules and is elaborated with comments at each necessary point to make it readable and understandable.

During the making of this program, the errors and mistakes were found out by using the **Debugging** available in codeblocks. In case of any warning or error, I checked each statement of the code through line by line execution in order to see its affect and figure out the possible flaws and mistakes.

MODULES OF SONGS

The code starts with the structure that is responsible for the input of the string data as well as the integer data in the in program.

```
typedef struct songs
{
    char name;
    char artist;
    char album;
    char genre;
    char year;
    char duration;
    char rating;
}s;
```

Following the data structure program basically consists of 8 functions that perform all different sorts of operation.

1. VOID CLRSCR()
2. VOID DELAY(INT MILLISEC)
3. VOID SONG_ENTRY()
4. VOID DISPLAY_DATABASE()
5. VOID NEW_DATABASE()
6. INT SONG_SEARCH(INT CHOICE)
7. VOID POPULAR_SONGS()
8. VOID SONG_RECOMMENDATION()

1.VOID CLRSCR()

I made this function in order to avoid the use of system(‘cls’) and replace it with an alternative that looked simpler to me.

2. VOID DELAY(INT MILLISEC)

This is a very short yet very interesting and practical function in the code. By using the clock_t which can only be used in the presence of <time.h> header file, we are able to count the time elapsed since the opening of program. Using that we can control the time of different outputs and speed of the loops and make things look different from the usual output style. This function has basically has pleasing purposes in the program.

3. **VOID SONG_ENTRY()**

This function is used to enter songs into the existing database. The function first opens the file in "**a+"** mode and checks its status. This function sequentially asks for all the parameters of songs to be entered into the file by using a repetitive loop and once all the parameters have been entered it puts a new line character (\n) which is of great practicality during traversing the code.

When we are taking input from the user in this function, in case if user enters different cases of alphabets at the same time we convert each input to lowercase by using *tolower()* operator and hence storing all the inputs in lowercase. This makes the whole data in the file to be in the same format hence makes it easier to traverse.

4.DISPLAY_DATABASE()

This is a small function with big practically. It opens the existing file in "**r"** mode and checks if the file contains some data of not. In case we have data in the file it read the *FILEPOINTER* to the *EOF* and prints each and every word in the file on the screen.

The part that makes it more interesting is the use of *VOID DELAY(INT MILLISEC)* function which controls the speed at which the data is printed on the screen. In the program we have assigned in a value **30 milli seconds,** which means a letter is printed on console screen after every 30ms interval.

5. VOID NEW_DATABASE()

This function is a special option operation that is available in the program. It is capable to making a new file and insert data in it. Just like the *SONG_ENTRY()* function this function also opens database in read mode and checks if it already exists or not. In case of its existence already the function terminates and stops from proceeding any further. Therefore, it is mentioned in parenthesis in the *MAIN MENU* of the program that only use this function if database does not exist already.

In case we have no file already, the function proceeds to taking inputs for the parameters of songs from the user, which are also based on the same principle as *SONG_ENTRY()* function. User enters characters which are then stored into the *STRUCTURE* using getchar() and the input for the parameter is terminated with newline character **\n**. The entered data is stored into file using *fputc()***.**

6. INT SONG_SEARCH(INT CHOICE)

This is probably the most complicated function of the program as it starts with lots of declarations and initializations. The basic concept of this function that we take two character pointer *mysearch* and *mydata* to read the data from the user and file respectively. In the start of function **0** memory is allocated to both of the pointers, but as we take input from the user through loop we increment its allocated memory using

the ***realloc***. The major advantage of using ***realloc*** is that it append the new data into the existing string. Once the data is entered by the user the string is given a end of line character **\0**.

On the other hand *mydata* reads the data from the file that has been opened and reads until it reaches the **\n**, once it gets there, it is also closed with **\0** and the two obtained strings are compared using ***strstr*** built-in function, which in case of true is printed on console screen, other wise the process is repeated until *mydata* reaches the ***EOF***.

We also used library function ***fflush(stdin)*** because the \n from the last scanf was causing issue on the output window.

In the end all the pointers are freed.

7.VOID POPULAR_SONGS()

This function is part the special extension that was given to my homework. This program uses the same concept of ***string manipulation*** as the ***SONG_SEARCH*** function. But the most interesting addition comes in the form of rating.

In this program we have given each song a rating ranging from **0.0-5.0**. The higher the rating the more popular is the song. In order to sort out the highest rated songs, I used an algorithm with the help of ***while loop*** which goes from ***5.0 to 0.0*** and is decremented at a rate of ***0.1***. The rating is basically taken as ***float*** because of its obvious nature and later it is converted into string using ***sprintf()***. At each distinct rating the, it is compared with ***mydata*** as in the previous function to find highest rated songs.

But the other part is that the number of popular songs should be random. For this objective we used another operator of **<*time.h*>**, rand() operator. To ensure that it gives distinct value and every iteration we used ***srand(time(NULL))*** which counts the seconds elapsed since ***1st January,1971*** so it's a new number at every interval. And for this function I decided I am going to have a maximum of **3** random songs, so we

```
srand(time(NULL));              /**< this logic with srand(),time() and rand() are used to
tim=rand()%4;
if(tim==0)
   {
       tim++;
   }
   printf("\n\n\t\t\tSearching");
   for(int sec=1000;sec>100;sec=sec-300)
{
    delay(sec);
    printf(".");
}
    while((n<=5.0) && (n>=0.0))         /**< ratings from 5.0 to 0.0 */
{
    num=n;
    sprintf(rat,"%f",num);              /**<the rating in float is converted to string */
    rat[3]='\0';
```

took ***modulus*** of rand with 4, rand()%4, and iterated the function the remainder number of times. In case the remainder becomes **0**, it is assigned a value **1.**

8.VOID SONG_RECOMMENDATION()

This function is a hybrid of the last two functions, in the previous function we compared two strings but for this function we compare 3 strings at the same time to get the desired result.

This is the 2nd part of specification extension to the homework. The goal of this function is to display the highest rated song of a given genre. For the rating part we use the previous function as it is and we take genre from the user as input and we use the part of ***SONG_SEARCH(INT CHOICE)*** function to accomplish this. This function prints the one highest rated song of the given genre.

```
if((strstr(mydata,mysearch)!=NULL) && (strstr(mydata,mygenre)!=NULL) && (decision==0))
```