ENEE 150: Intermediate Programming Concepts for Engineers

Spring 2019 **Handout #24**

Project #3: On-Line Music Server
Due Date: Apr 15, 11:59p.m.

In this project, you will build a program that implements a simple on-line music server. Your program will maintain information about music albums that customers can access. Your program will also permit users to manage accounts within the on-line server, and to create playlists containing their favorite songs. Because you will not be told in advance how many albums there will be, how many users there will be, nor how large the users' playlists will be, you must employ dynamic data structures to implement your project.

1 Overview

Your on-line music server will operate in two modes: initialization and transaction handling. Your program should enter the initialization mode upon startup. During intialization, your program will create the music albums that your program will manage. Once initialized, your program should then enter the transaction handling mode. In this mode, your program will receive user transactions, each requesting your server to perform different operations. These include opening a new user account, adding to the playlist of an existing user account, closing an existing user account, and printing information about both music albums and user accounts. Your program will receive all information for both intialization and transaction handling through input files. Furthermore, all information provided back to users will be through standard output.

2 Data Structures

Your program will store three types of information: music albums, user accounts, and user playlists. This section describes the data structures you should use to store these different types of information. All of the data structures described in this section must be allocated dynamically.

2.1 Music Albums

Your program should store the music album information in an array of structures, where each structure in the array stores the information associated with a single album. The top and lower right portions of Figure 1 illustrate this data structure. Each music album is identified by a unique ID which corresponds to the album structure's index in the album array. Each album structure itself contains three fields. The "num_tracks" field is an

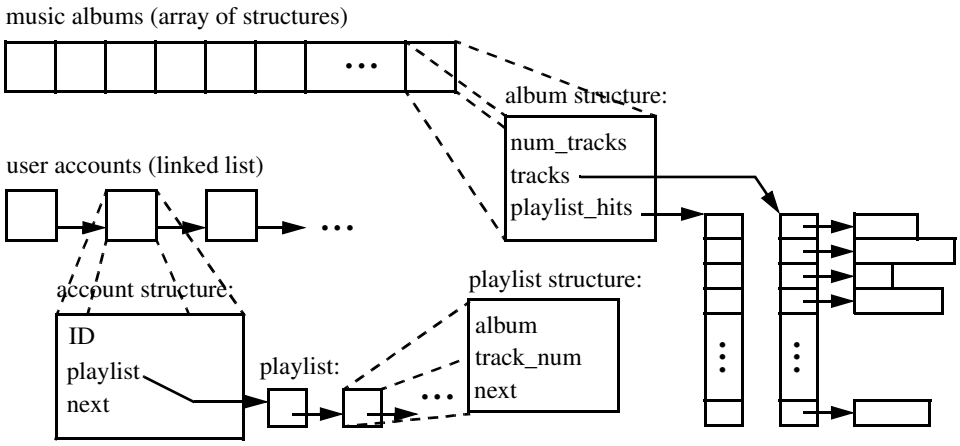

Figure 1: Recommended data structures for implementing the on-line music server.

integer that specifies the number of tracks contained in the album. The "tracks" field is a pointer to a pointer array that holds the title of each track. The pointer array contains one pointer for each track in the album, so it has "num_tracks" entries. Each pointer in the pointer array is a pointer to a character array that holds a string containing the track title. Lastly, the "playlist_hits" field is a pointer to an array of integers, one per track (so there are "num_tracks" integers in this array). Each integer in this array records the number of users that have included the corresponding track in their playlists.

2.2 User Accounts

Your program should store the user accounts information in a linked list, where each link node stores the information associated with a single user. The lower left portion of Figure 1 illustrates this data structure. Each user account structure contains three fields. The "ID" field is an integer value that holds a unique user ID. The "playlist" field is a head pointer for a linked list that stores the user's playlist information (see Section 2.3). Lastly, the "next" field is a self-referencing pointer that points to the next link node in the linked list.

2.3 Playlists

For each user account, your program will maintain a playlist that stores the user's favorite songs from the albums managed by the on-line music server (each playlist entry is an album ID and track number). There should be a playlist for every user which is a linked list whose head pointer is the "playlist" field in the user account structure. Each link node is a playlist structure that contains 3 fields. First, the "album" field is an integer that specifies the album ID for the playlist entry. Second, the "track_num" field is an integer value that specifies the track number in the album containing the song on the playlist. Finally, the "next" field is a self-referencing pointer that points to the next link node in

the linked list.

3 Input Files

Your on-line music server will implement two modes of operation: initialization and transaction handling. During each mode of operation, your program should read information from a separate input file. An "album" file contains all the information needed during the initialization mode. A "transaction" file contains all the information needed during the transaction handling mode.

We have provided 5 test cases for you to drive your program, with each test case providing both an album and transaction file. The test case files are named "test1.album," "test1.xact," "test2.album," "test2.xact," "test3.album," "test3.xact," "test4.album," "test4.xact," "test5.album," and "test5.xact." We have also provided the output that your program should give in response to these 5 test cases in the files "test1.out," "test2.out," "test3.out," "test4.out," and "test5.out." All of these files can be downloaded from the course web site (follow the hyperlink labeled "Project 3 Files").

The names of the album and transaction files your program will process will be provided to your program as command line arguments. For example, to run your project on the first test case, you should type (assuming your program is called "pr3"):

```
pr3 test1.album test1.xact
```

4 Server Operation

4.1 Initialization

Upon startup, your program should build the array of structures described in Section 2.1 that contains the music album information. To build this data structure, you need to know how many albums there will be (so that you can malloc the array of structs), and for each album, you need to know the number of tracks in that album (so that you can malloc the tracks and playlist_hits arrays), and the title of each track. All of this information is provided in the albums file. Your program should open this file, and parse it to read all the music albums information.

The format of the albums file is as follows. The file begins with a single integer value that specifies the number of albums you should create. This is followed by information for each album. Each album's information is specified as follows. First, a single integer value is provided that specifies the number of tracks in the album. Then, for each track in the album, there is a single line that specifies a string containing the track title. In each track title line, there is first an integer value that specifies the length for the track title string

(excluding the NULL character that would terminate the string), and then there is the string itself. So, an album in the albums file has the following format:

```
number of tracks (N)
length1 string1
length2 string2
length3 string3
.
.
.
lengthN stringN
```

Note, when creating the "playlist_hits" array for an album, you should initialize all the array values to zero.

4.2 Transaction Handling

After creating the music albums data structure, your program should enter the transaction handling mode. All the user transaction requests are contained in the transaction file. Your program should open this file, and parse it to read all the transaction requests. Each transaction request appears one after another in the file. A transaction request begins with a single integer value that specifies the transaction ID, and is followed by zero or more integer values that provide additional information needed by the transaction. After reading each transaction request, you should perform the operation associated with the transaction, and then read the next transaction request.

In total, there are 6 different types of transactions that your program must handle: print_album, open_account, print_account, add_playlist, close_account, and terminate.

4.2.1 Print Album

This transaction is specified when the transaction ID is equal to "1". After the transaction ID, there is a single integer value that specifies an album ID. When your program encounters a print album transaction, your program should print the music album associated with the specified album ID. First, print "ALBUM" followed by the album ID. Then, print the information for each track in the album on a separate line. In particular, you should print the "playlist_hit" value for the track, followed by a colon, followed by the track title. Here's an example of printing an album (from test1.out):

```
ALBUM 0
  0: So You Want to Be a Rock 'N' Roll Star
  0: Have You Seen Her Face
  0: C.T.A. - 102
```

```
0: Renaissance Fair
0: Time Between
0: Everybody's Been Burned
0: Thoughts and Words
0: Mind Gardens
0: My Back Pages
0: The Girl with No Name
0: Why
0: It Happens Each Day
0: Don't Make Waves
0: My Back Pages
0: Mind Gardens
0: Lady Friend
0: Old John Robertson
```

When implementing your code for the print album transaction, use the first test case to test your code. This test case initializes your server with 10 albums (in test1.album), and then sends 10 print album transaction requests to your server (in test1.xact), one for each album you created. See the test1.out file for the proper output.

4.2.2 Open Account

This transaction is specified when the transaction ID is equal to "2". After the transaction ID, there is a single integer value that specifies a user ID. When your program encounters an open account transaction, your program should create a new user account with the specified user ID. To create the new user account, malloc a new account structure, initialize the ID field with the specified user ID, and initialize the playlist field to NULL. Then, insert the new user account into the user accounts linked list. **When creating a new user, you should always insert the new user at the head of the user accounts linked list.**

4.2.3 Print Account

This transaction is specified when the transaction ID is equal to "3". After the transaction ID, there is a single integer value that specifies a user ID. When your program encounters a print account transaction, your program should find the user account with the specified user ID, and print the account's information. First, print "ACCOUNT" followed by the user ID. Then, if the playlist for the user account is empty, print "EMPTY PLAYLIST". Otherwise, print the user account's playlist, placing each playlist track on a separate line. For each track in the playlist, print "ALBUM" and then the ID of the album that contains the track, followed by a colon, followed by the track title. You should print the playlist entries in the order that they appear in the user's "playlist" linked list. Here's an example of printing an account with an empty playlist (from test2.out):

```
ACCOUNT 6838
EMPTY PLAYLIST
```

Here's an example of printing an account with a playlist (from test3.out; see Section 4.2.4 for details on how to add playlists):

```
ACCOUNT 6838
  ALBUM 8:  Did'n I?
  ALBUM 5:  Horizon [Moore]
  ALBUM 7:  Dirty South - The End
  ALBUM 9:  Euphoria / One In A Million
  ALBUM 6:  East 17 - House Of Love
  ALBUM 7:  Ercola Vs Heikki L - Deep At Night (Adam K & Soha Remix)
  ALBUM 0:  Why
  ALBUM 3:  We've Only Just Begun
  ALBUM 3:  Always
  ALBUM 6:  Yello - Jungle Bill
  ALBUM 8:  I Wanna Know
  ALBUM 1:  My Love is a Flower
  ALBUM 4:  New York Connection
  ALBUM 5:  Dance Of The Emperor's Clouds [Gascoigne]
  ALBUM 6:  Cathy Dennis - You Lied To Me
  ALBUM 5:  All At Sea Minor [Bach arr. Myers]
  ALBUM 2:  The Temptations - My Girl
  ALBUM 7:  Alex Guesta - The Groove Of Love (Get-Far Mix)
  ALBUM 3:  The Twelfth of Never
  ALBUM 8:  In Da Club
```

When implementing your code for the print account transaction, use the second test case to test your code. This test case initializes your server with 1 album (in test2.album), and then sends 5 open account transaction requests followed by 5 print account transaction requests to your server (in test2.xact). See the test2.out file for the proper output.

4.2.4 Add Playlist

This transaction is specified when the transaction ID is equal to "4". After the transaction ID, there are 3 integer values that specify a user ID, an album ID, and a track number. When your program encounters an add playlist transaction, your program should find the user account with the specified user ID, and add a new playlist entry for the user that refers to the album and track specified by the album ID and track number. To create the new playlist entry, malloc a new playlist structure, initialize the album and track_num fields with the provided album ID and track number. And then insert the new playlist struct into the user's "playlist" linked list. **When inserting playlist entries, you must**

insert at the tail of the linked list. Finally, increment the "playlist_hits" value for the corresponding track in the albums structure.

When implementing your code for the add playlist transaction, use the third test case to test your code. This test case initializes your server with 10 albums (in test3.album), then sends a single open account transaction, then sends 20 add playlist transactions, then sends 1 print account transaction, and finally sends 10 print album transactions to your server (one for each created album). See the test3.out file for the proper output.

4.2.5 Close Account

This transaction is specified when the transaction ID is equal to "5". After the transaction ID, there is a single integer value that specifies a user ID. When your program encounters a close account transaction, your program should find the user account with the specified user ID, and remove it from the user accounts linked list. Your program should also traverse the playlist linked list for the account, and remove every link node from the playlist linked list. For each playlist link node, you should find the album and track specified by the playlist link node's album ID and track number fields, and decrement the corresponding playlist_hits value. Finally, you should free the user account structure and all the playlist structures associated with the closed account.

When implementing your code for the close account transaction, use the fourth test case to test your code. This test case initializes your server with 10 albums (in test4.album), then sends a single open account transaction, then sends 20 add playlist transactions, then sends 10 print album transactions (one for each created album), then sends a single close account transaction, and finally sends 10 more print album transactions. See the test4.out file for the proper output.

4.2.6 Terminate

This transaction is specified when the transaction ID is equal to "6". There are no other values associated with this transaction. Your program should exit when it encounters the terminate transaction. All the test cases provided on the course web site end with a terminate transaction.

When you have finished implementing all the transactions, try your program on the fifth test case. This test case initializes your server with 10 albums (in test5.album), and sends 100 randomly generated transactions to your server. See the test5.out file for the proper output.