CSE225 Data Structures

PROJECT #2 (Due to: November 30, 2013, Saturday, 08:00 a.m.)

This project is a programming assignment in C which builds a music-album based on the user's preferences. This program will let the user add, remove, update his/her music albums in his/her music-box.In this project you are expected to make your implementation by using ***Binary Search Trees***.

The data structure that you use is as detailed below:

1. The album title,
2. The year the album is/was released,
3. The name of the singer of the album,
4. A list of the songs in the album. For each song in the album, you should keep:
   - The name of the song
   - The length (seconds) of the song

In your implementation you MUST keep the album entries in a binary search tree by release year of the album. Then, to keep the songs of each album, you should create a binary search tree in which the songs are kept by their names (e.g., if you have five albums in your binary search tree, you will have five more binary search trees: one for the song list of each album.). You may define your binary search tree such that for a node, smaller valued items are kept in the left subtree of the node and greater or equal valued items are kept in its right subtree.

Your system will have the following functionalities; the details of these functionalities are given below:

1. Add an album
2. Remove an album
3. Show the list of albums
4. Show detailed information about a particular album
5. Add a song to the song list of an album
6. Remove a song from the song list of a album
7.Show the list of all songs
8. Query the albums which are/were released between particular years
9. Query the songs whose lengths are in a particular scope

1. ***Add an album:*** This function adds an album to the *Album-Binary-Search-Tree* for a given album whose title, singer's name, and release year are specified as parameters. In this function, the song list is not specified; the songs listed in the album will be added later. In this system, the titles of the albums are unique. Thus, if the user attempts to add an album with an existing title, you should not allow this operation and give a warning message.

2. ***Remove an album:*** This function removes an album from *Album-Binary-Search-Tree*, whose title is specified as a parameter. If this album does not exist in the *Album-*

Binary-Search-Tree (i.e., if there is no album with the specified title), you should not allow this operation and give a warning message.

3. ***Show the Album-Binary-Search-Tree:*** You should list all album entries in the *Album-Binary-Search-Tree* on the screen in the following format. If the *Album-Binary-Search-Tree* does not contain any album, you should display ---none---.

   Title, release year, singer’s name (for the 1st album)
   Title, release year, singer's name (for the 2nd album)
   Title, release year, singer's name (for the 3rd album)
   . . .

4. ***Show detailed information about a particular album:*** You should display all of the information about an album whose title is specified as a parameter. The output should be in the following format. If the album with a specified title does not exist in the *Album-Binary-Search-Tree*, you should display ---none--- after the title.

   Title
   Release year
   Singer's name
   Song name, length of song (for the 1st song)
   Song name, length of song (for the 2nd song)
   Song name, length of song (for the 3rd song)
   . . .

5. **Add a song:** This function adds a song to the Song-Binary-Search-Tree, of an album. For that, the title of the album to which the song is added, the name of song, and the length (in seconds) of song are specified as parameters. In this function, you should consider the following issues:
   • If the album with a specified title does not exist in the *Album-Binary-Search-Tree*, you should not allow the operation and give a warning message.
   • In this system, all names of the songs are unique. Thus, if the user attempts to add a song with an existing name, you should not perform the operation and give a warning message.
   - The Song-Binary-Search-Tree, should be in alphabetically ascending order according to the song name.

6. **Remove a song:** This function removes a song from the Song-Binary-Search-Tree, of an album. For that, the title of the album from which the song is removed and the name of song are specified as parameters. If the album with a specified title does not exist in the Album-Binary-Search-Tree, you should not allow the operation and give a warning message. Similarly, if the song is not in the Song-Binary-Search-Tree, of the specified album, you should not allow the operation and give a warning message.

7. **Show the List of All Songs:** This function lists all the songs in all existent Song-Binary-Search-Trees. Note that if there is no songs by any albums, you should display ---none--- .

   The output should be in the following format:

   Song name, length of song (for the 1st song)
   Song name, length of song (for the 2nd song)
   Song name, length of song (for the 3rd song)
   . . .

8. ***Query the albums which were released between particular years***: You should list all albums produced between the particular years which are specified as parameters. The output should be in the following format. Note that if there is no album produced between in those years, you should display ---none---.

   Album title, Singer Name, release year (for the 1st album)
   Album title, Singer Name, release year (for the 2nd album)
   Album title, Singer Name ,release year (for the 3rd album)
   . . .

9. ***Query the songs whose lengths are in a particular scope***: You should list all songs whose lengths are in a particular scope those scope levels (upper bound and lower bound) are specified as parameters. The output should include the name of the song, the length of the song and the singer of the song, and should be in the following format. Note that if there is no songs in that particular length-scope then you should display ---none---

   Song Name, the length of the song, singer name (for the 1st song)
   Song Name, the length of the song, singer name (for the 2nd song)
   Song Name, the length of the song, singer name (for the 3rd song)
   Song Name, the length of the song, singer name (for the 4th song)
   . . .

Below is the required part of the MusicBoxTree.c file that you must write in this project. The name of the file must be MusicBoxTree, and must include these functions.

```
addAlbum( … albumTitle, … singerName, ...releaseYear );
removeAlbum( …albumTitle );
showAllAlbums();
showAlbumDetails( … albumTitle );
addSong( … albumTitle, … songName, …songLength );
removeSong( … albumTitle, … songName );
showListOfAllSongs ();
showAlbumsInBetweenParticularYears ( …
LowerBoundaryYear,UpperBoundaryYear );
showSongsWithParticularLength(…LowerBoundary,UpperBoundary );
```

Make sure that the name of the file is MusicBoxTree, and the required functions are defined as shown above.

After running your program there should be a main menu in your terminal which includes above functionalities as menu-items(like in your first project) so that user can examine each of them.
In your demo, we will run your program by **selecting each of menu-options.** We want to see if they are **working correctly or not**, **and your demo grade will be calculated based on number of your functions which are working correctly.**

Of course, other questions based on your implementation and coding structure will be asked you. These questions will be those kinds of questions which could be answered by only the students who really implement his/her project.

If your file could not be compiled then you will get zero, unfortunately.
The main goal of this project is to be familiar with binary-serach-trees. So, if you use data structure other than binary- serach-trees then you will get zero, unfortunately.

In this project you are expected to develop an algorithm that is capable of finding a solution to the above problem and ***implement this algorithm in ANSI C that runs under UNIX***. (You can also use cygwin or wagrant in order to compile and run your codes.)

You are responsible for demonstrating your program to your TA Berna Altinel on the scheduled day that will be announced later.

CODE SUBMISSION:

As you did for your first project, again you need to submit this project into your bitbucket repository.

(This explanation is for new comer students)

Before submitting your project:

1. You should take an account from https://bitbucket.org/.
2. You should create a **private** repository with the same name of your project.
3.You should add **Assoc. Prof. Borahan Tumer , Asst. Prof. Fatma Ergin and TA Berna Altınel into your project and** and give them access permission to your project. **YOU SHOULD NOT GIVE OTHER PEOPLE ANY ACCESS PERMISSION.THIS IS YOUR RESPONSIBILITY TO MAKE YOUR PROJECT PRIVATE.**
4. While you are implementing your projects, you are expected to commit your codes into your bitbucket account at each milestones.
5. **Your any submission/commit after the project submission due date, will not taken into consideration.**

Please keep this in mind and **promptly start working on your projects**!

You are required to exhibit an ***individual effort*** on this project. Any potential violation of this rule will lead everyone involved to **failing from the course** and necessary disciplinary actions will be taken.

Good luck!!!