

## AVL Music Database

For this project you will implement an AVL tree template and use it to create a database for a music collection. Logically, each node of the AVL tree will store a record containing information about a particular recording artist:

- artist name (a string of unspecified length, duplicate names are not allowed)
- a dynamic list of recordings by that artist; For each recording you will store a title, format and year.

The AVL tree will use the artist name as its primary key field. All string comparisons will be case-sensitive.

The AVL tree index will support the usual (instrumented) search, insert and delete operations.

On startup, the program will read a text file containing recording information, and build an AVL tree containing nodes for those recordings. Next, the program will read a command file and carry out the commands, writing any output to a log file.

### Data Structures:

The primary data structures element of this project is an AVL tree, as described in the course notes. Your implementation is under the following specific requirements:

- You must encapsulate the AVL tree and its nodes as C++ templates.
- The data that is stored in each node must be encapsulated in a C++ class.
- The behavior of the AVL tree must conform to the description in the course notes. The only important distinction between the implementation in the notes and that in Kruse/Ryba is whether deletion of a node with two children involves a swap from the right subtree or the left subtree. Your implementation should always use the minimum value in the right subtree in that case.
- Given the nature of this project, no two distinct tree nodes will contain duplicate keys. That will require a modification to the insert logic.
- Since the number of recordings by an individual artist is impossible to predict, the list of recordings should be dynamic. You may use a linked list, a resizable array, an STL vector object, or an STL list object. Whichever alternative you select, your list should be encapsulated as a class (or, better, as a template). If you've never used an STL component, this would be a good opportunity to start.
- For testing, your AVL tree should have the ability to display itself to a specified output stream, as described in the notes. If you expect to receive help, be sure that your display function conforms to the formatting described in the course notes.

Your design must make appropriate use of classes. The specification may imply the existence of additional classes besides those involved in the implementation of the AVL tree. Aside from binary nodes used only within an encapsulating class, data members of classes must be private.

The use of inheritance in this project is up to you; arguments can be made in favor of, and against, using inheritance for the AVL template or the node template.

## Program Invocation:

Your program **must** take the names of the input and output files from the command line — failure to do this will irritate the person for whom you will demo your project. The program will be invoked as:

```
MusicDB <init file name> <command file name> <log file name>
```

If either of the specified input files does not exist, the program should print an appropriate error message and either exit or prompt the user for a correction.

## Text File Description:

The initialization file is just that, a file containing data for an unknown number of recordings. Each line of the initialization file will contain data for a single recording, formatted as:

```
<artist name><tab><title><tab><format><tab><year><newline>
```

Note that it is possible (likely, really) that the same artist name will be associated with more than one title. If the artist name already occurs in the AVL tree, then you will simply add the current title to the list for that artist. However, duplicate titles (for the same artist) are not allowed; if a duplicate title is detected then no entry is added. Here is a short sample initialization file:

Clapton, Eric	Crossroads Disk 1	CD	1988
Clapton, Eric	From the Cradle	CD	1994
Seeger, Bob	Greatest Hits	CD	1994
Derek and the Dominos	Layla	LP	1970
Allman Brothers Band	The Fillmore Concerts	CD	1992
B. B. King	Riding with the King	CD	2000
Clapton, Eric	Riding with the King	CD	2000

## Driver File Description:

Lines beginning with a semicolon ( ` ; ` ) character are comments; your program will ignore comments. An arbitrary number of comment lines may occur in the input file.

Each non-comment line of the command file will specify one of the commands described below. Each line consists of a sequence of “tokens” which will be separated by single tab characters. A newline character will immediately follow the final “token” on each line.

**artist** <name>

This causes the AVL tree to search for the given artist name. If the artist name is not found, an appropriate error message should be written to the log file. If the artist name is found, then the level of the node containing it should be logged.

**list** <name>

This command is similar to the `artist` command, except that the logged data consists of a listing of the titles by that artist. If the artist name is not found, log an error message.

**count** <name>

This logs the number of titles, if any, that are stored for the named artist. **If the named artist is not found, an error message should be logged.**

**title** <title>

This causes the AVL tree to search for recordings that match the given title. If none are found, an error message should be logged. Otherwise, a list of the matching artist names and years should be logged.

**remove** <name>

This causes the AVL tree to search for a node containing the given artist name. If none is found, an error message should be logged. Otherwise, that node should be deleted from the AVL tree, and a confirming message should be logged.

**display**

This causes the AVL tree to be displayed to the log file. The display should include the key field (artist name) and the balance factor (-, /, or \) for each node. You may use either an inorder traversal or a modification that prints the right side first.

**exit**

This causes the indexing application to terminate. The command file is guaranteed to end with an exit command.

**Legend:** in the commands above:

<name>	a string naming an artist, contains no tabs
<title>	a string naming a recording, contains no tabs

You may assume that the input file will conform to the given syntax, so syntactic error checking is not required. If an error occurs during the parsing of the command file, there's an error in your code. However, your program should still attempt to recover, by "flushing" the current command and proceeding to the next input line.

## Log File Description:

Since this assignment will be graded by TAs, rather than the Curator, the format of the output is left up to you. Of course, your output should be clear, concise, well labeled, and correct. The first two lines should contain your name, course (CS 2604), and project title.

After the initial AVL tree is built from the initialization file, you should print a statistical summary of the tree's properties, including the total number of tree nodes and the height of the tree.

The remainder of the log file output should come directly from your processing of the command file. You are required to echo each command that you process to the log file so that it's easy to determine which command each section of your output corresponds to.

## Submitting Your Program:

You will submit a zipped file containing your project to the Curator System (read the *Student Guide*), and it will be archived until you demo it for one of the GTAs. Instructions for submitting are contained in the *Student Guide*. You will find a list of the required contents for the zipped file on the course website. Follow the instructions there carefully; it is very common for students to suffer a loss of points (often major) because they failed to include the specified items.

Be very careful to include all the necessary source code files. It is amazingly common for students to omit required header or cpp files. In such a case, it is obviously impossible to perform a test of the submitted program unless the student is allowed to supply the missing files. When that happens, to be fair to other students, we must assess the late penalty that would apply at the time of the demo.

You will be allowed up to five submissions for this assignment, in case you need to correct mistakes. Test your program thoroughly before submitting it. If you discover an error you may fix it and make another submission. Your last submission will be graded, so fixing an error after the due date will result in a late penalty.

The submission client can be found at: <http://spasm.cs.vt.edu:8080/curator/>

## Programming Standards:

The GTAs will be carefully evaluating your source code on this assignment for programming style, so you should observe good practice. See the Programming Standards page on the course website for specific requirements that should be observed in this course.

## Evaluation:

You will schedule a demo with your assigned GTA. At the demo, you will download your submitted project, perform a build, and run your program on the supplied test data. The GTA will evaluate the correctness of your results. In addition, the GTA will evaluate your project for good internal documentation and software engineering practice.

Here is an estimate of how much weight will be given to each of the factors that the GTA will consider:

Factor	Est, Weight
<code>artist</code> command	15%
<code>list</code> command	10%
<code>count</code> command	5%
<code>title</code> command	10%
<code>remove</code> command	10%
AVL and node template designs <ul style="list-style-type: none"> <li>▪ use of <code>public/private</code> access</li> <li>▪ data members</li> <li>▪ dynamic memory management, copy issues</li> </ul>	20%
Data class designs <ul style="list-style-type: none"> <li>▪ use of <code>string</code> type</li> <li>▪ overloading of relational operators</li> <li>▪ dynamic list for locations</li> </ul>	15%
Quality of internal documentation	15%
Other	5%

This has some implications for your strategy in developing this project. AVL deletion is involved only in the handling of the `remove` command, so if you don't implement that at all you will lose 10 points, plus (possibly) some additional points for incorrect comparison stats on some `artist` commands.

If you cannot get a working AVL tree at all, and base your implementation on a BST instead, then you will probably lose about half of the points for the `artist` command, some of the points for the `remove` command, and all of points for design. In any case, an implementation based upon a pure BST will receive a score no higher than 60 points.

## Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided with the earlier project specifications in the header comment for your main source code file.