Debugging and Design

gdb and continuous design modification was required when working through this assignment. gdb helped in debugging memory leaks and semantically checking the program for behavioral correctness. Running the program and running the stack backtrace helped in finding the memory leaks and segmentation faults. Further stepping through main to find the semantic errors in the program was considerably more difficult in order to find the break points to set. A particular area where gdb was used was when the insertion was being done in the hash table and graph to that each nodes pointer either pointed to another vertex in the case of a graph or in the case of the hash table that each node pointed to a different song. This process was used to modify how the data was sent into the data structures so that it reduced the number of memory leaks and efficiency. gdb was used to step into some of the insert and remove functions of the data structures to find whether a memory leak had occured during the operation. In particular the commands used were the step and next functions. The code breaking was found with using the debugger in the graphical code interface using the tui command. The memory leaks in the program were checked using valgrind, since gdb has no functionality to do so.

The program is composed of several classes from a bottom up design there classes are composed of a song class which holds the data for a song such as the artist name, the song name and the type of genres contained in the song, further the node encapsulates the song, which may for a linear linked list, the song may also form a linear linked list itself as well since it does contain a next pointer thus involving self similarity. This class and its associated members forms the basis which other classes use to add to their own data, operator overloading is used to add, remove or copy song objects throughout the data structures. Operator overloading is used throughout the program either as a display, assignment and/or removal and insertion. Some operators are not overloaded because the complexity of the overloading breaks some of the code for example the insertions in the graph is not overloaded due to the synchronous use of the edge to point to other graph nodes. However both the playlist, song, hash table, node all use the operator overloading in their basic use. The display function in the hash table is another place where operator overloading is not done and also removing a particular artist from a hash table. In hindsight the hash table doesn't require a remove function but has one due to requirements. Each data structure has the full implementations of insert, retrieve, delete, display and etc. This was done as program requirements most program requirements are satisfied, some of these implementations take place as operator overloading, however the graph class does not do operator overloading. In conclusion operator overloading is used in the program however the trade-off between complexity and code functionality is used to decide where it was and was not implemented.

1

How well did the data structures perform for the assigned application?
The data strucutres performed the operations of the assigned application mostly well with regards to efficieny and object oriented design. The design of the classes were by the objects that are instrinsic to the assignment for example, storing the song objects in their respective data structures based on artist or genre each differed in responsibility and searching despite having the song object stored in each node of the structure as encapsulated design. In this design most of the classes had very similar linked structures but each differed in some form of design fundamentally. Although the complexity is reduced somewhat using OOP the modularity of the program is decreased.

2

Would a different data structure work better? If so, which one and why.
In the case of the program problem the data structures chosen fit the task well. A alternative data structure that can be used in place of a graph would either be a tree or another hash table. Each of these has its own advantages with their respective running times. However the interconnectedness of the graph is not something that is found in many other data structures. For example doing a depth first search based on a name of a song and the genre may not be available in the tree data structures or hash table. Since the graph does implement a linear linked list with the increasing number of songs and number of vertices with

form continued on next page...

more similar genres the insertion or retrieval would run slower as more songs are added to the database. The memory usage on the stack would increase quadratically as more songs are added. In the case of adding more data to the hash table or the trees copies of the data are added as more nodes are made, the copies are added to the genre nodes. In conclusion the data structure that fits the program design well is the graph.

3

What was efficient about this program design?
The efficiency of the design lies in the object oriented design of the program. The responsibilities are characterized as encapsulation or responsibility of each object. Each class working with data structures do implement the remove, insert and other data structures related functions. Although the code reusage is extensive throughout the program the data structures may not have much in common to form a heirarchy. The relationships as encapsulation served to simplify the code and reduce the complexity in design which contributed towards the efficiency of the design. Further using the tables both reduced the linear searching and increased run time efficiency.

4

What was not efficient about this program design?
The inefficiency of the program lies in the code reusage. For example both the node and song pointers have a self similar strucutre(the next pointer). In this case bother can be treated as a linear linked list. However the only data structure that should use this structure is the node and playlist shoudl be the nodes in the hash table. Further the hard coded constants for the data structures also add to the inefficiency. Some of the constants may not relect the optimized data structure and add to the operations like traversing, assignment, etc. Another glaring inefficiency of the program is how the graph operates for example if a song is added to the graph the number of genres is the number of copies that are added to the graph, a song with three genres allocated three times to each vertex of the genre. With the a large database this may cause stack-overflow.

5

What would you do differently?
Given more time, I would modify the hash table to be a linear linked lists of playlist instead of doing nodes with another linked list class. This would be a better design given the problem. I would also chang the hard coded constants for some of the data strcutures to better represent the size of the problem using the file structure.