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 BST to check the balance of the tree. This processed was used to modify how the data was sent into the tree in particular modification of the map function to be more random inorder to match the data. 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 term class which holds the data that comprises a term. This may be the number of credits, name of term and etc. This class and its associated members form a base class heriarchy from which the binary tree node derives from. Further the design contains a linear linked list node class where dynamic binding is used. Dynamic binding is used mostly in the course classes like general, required, core and non-core classes. The base class or the general class provides a heirarchy where dynamic binding may be used. The design does lack a abstract base class with pure virtual functions this was mostly done out of time constraints after having designed the program erroneously to begin with. Some of the objects in the design use other objects that are contained as members to instantiate other member functions, which in this sense may not be an object oriented approach. For example once a list is made in the binary tree node class the functions like credits, gpa and requirement satisfaction is updated. 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 but the pure abstract base class function is not one of those.

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, the binary tree node object and linked list object each had their own responsibilty despite being encapsulated. In this program design the classes each had their own responsibilities, for example the general was strcutured as a node representing a linear linked list. The binary tree node class contained and derived from the term object. 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.
A data structure that would better accomplish this task with fewer lines of code and more efficientcly would either be a hash table to store the term and class data and a binary tree to store the requirements or a combination of balanced tree and an array. Each of these has a its own advantages with the hash table the semi-direct access and the efficiency at run time pay off with fewer chains (nodes in the linked list), the requirements which in this case are the courses that must be taken to be accepted into the program can easily be stored in the binary tree based on ascii value of each letter. The balanced tree with the combination of array balances out the memory usage and direct access while always ensuring a lower bounded logarithmic efficiency. Both of these approaches have their respective memory and run time pay offs but both are different data structures that can be replaced in the context of this assignment.

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 inheritance. Each class working with data structures do implement the remove, insert and other data structures related functions. The inheritance design is also very useful when code needs to be reused. The heirarchical relationships were between the different courses like required, not required and general classes. The relationships served to simplify the code and reduce the complexity in design which contributed towards the efficiency of the design. Further using the binary tree sometimes had a lower bound of logarithmic efficiency in run time, this may not however be the case (see ineficiency). Dynamic binding was used in the form of downcasting when displaying and deleting linked list nodes which represented courses.

4

What was not efficient about this program design?
The inefficiency of the design is prevalent in the relationship between the binary tree and the linear linked list, a array could have easily replaced the linear linked list. Further a balanced tree would be a better way to tackle the problem of how data is entered into the linked list. For example entering the data by term and using a integer value to support the insertion based on a integer value does result in a linked list list efficiency in remove, insert and display with the nodes being on and/or the other side. The design further depends on the requirments being stored in a linear linked list which form a linear sequence thus requiring algorithms withs a linear lower bound. The binary tree may have lower bounded logarithmic run time algorithms but this is not always the case since removing from the binary tree may result in a linear run time.

5

What would you do differently?
Given more time a balanced tree might be a better approach to solve this problem. Implementing a red black tree is would be a more superior aglorithm but with a little more memory compared to binary search tree. Further the task of keeping track of the requirements was done with a linear linked list. This approach can be changed by using an array or another tree of courses by modifying the heirarchy to extend a tree to be courses.