1

How well did the data structure perform for the assigned application?

The data strucutre comprises of a linear linked list, with a power object of struct type and a dynamically allocated array of hero object which are also struct type. The data strucutre consists of a constructor to initialize the head, temp and current pointers. Further the data structure consists of a load function that will read the data files and create a linear linked list for the power and hero objects. For the user interface a add hero, remove power and display functions are implented. The assigned application was to read a power and hero file and create the assigned data structure for the user to interact with, this function has an efficiency of $O(n^3)$. The user can select the display all of the powers and heros associated with the particular power or an alternative display function which will take an argument of a string of characters representing the power and match it to the characters. Both of these functions had a run time efficiency of $O(n^2)$. Both involved a sequential search and iteration through the hero objects array. Both of these objects perform the display task without any run time errors, however better data structures can be designed for efficiency purposes (see 2). For further usability the program also contains a add hero and remove power function, however these functions do not write back to file. The add hero function will add a hero object by asking the user for name which is limited to 33 characters and any alpha-numerical or symbol otherwise for example 7-up Man or λ Woman. Then an iterator will search the linked list to find the particular powers for the hero and add it to the array structure. Which involves a copy and delete (see program for more details). The remove power function involves a sequential search and going through the motion of dragging a temporary pointer to do the normal delete operations for a linear linked list.

2

Would a different data structure work better? If so, which one and why.

A tree with head pointer to power nodes might be an easier data structure to use for a database and problem like this. There is less memory overhaul and a depth first search might be an easier function to implement for a particular node. However for large tree structures this might be worse, an implementation would probably resolve this question. The children of the parent power objects can be the hero objects and the powers can be linked lists. However this still ends up as memory intensive linked list and requires heavy duty search algorithms.

3

What was efficient about this program design?

The efficiency in this design relies on the memory aspects of the data structure. Each power was a separate entity in the run time memory and not contiguous. However the downside of this can be the number of super heroes allocated to memory is contiguous and for a larger database this can be a problem.

4

What was not efficient about this program design?

The amout of memory for copying, assiging and interacting with the data structure was an issue. Sequential searches and loading for larger databases will eventually become an issue if these algorithms are used. Further the linked list might not be very generic and work with different databases. This projects scope was very limited.

5

What would you do differently?

In hindsight it would be better to write to files to add heros and their descriptions so that they can be used when the program is called again. Further the user interface could be designed better to interact with the user. More pointers could be allocated to find powers faster, for a alphabetically sorted linked list. The prompt for the user to enter a particular situation might match more than one super power, which will return a power that is nearest to the searching pointer.