CS 202 EFFIECIENCY AND GDB WRITE-UP FOR PROGRAM 1

BY TEJAS MENON; ASSIGNMENT BY PROFESSOR KARLA FANT

The data structure chosen for the current program was appropriately sufficient for performing a variety of possible tasks with a limited size galaxy and solar system. For example, an array of solar systems allowed for an easily implementable body of code with traversal techniques that were equally easy and a DLL of planets for each solar system allowed for resizing/sorting at runtime. However, such a simplistic solution for more complex operations, such as in a case where each solar system would require reordering/positioning/shifting, would have been difficult and inefficient to perform on an array. Additionally, as in a real world application, if the galaxy sizes chosen had to be several magnitudes larger, it would have been difficult to store such a large array contiguously in memory. An alternative would have been to store solar systems in a BST/balanced tree, arranged by their position in space or likelihood of life in a meaningful way. This would have made the process of accessing particular solar systems easier upon expansion of the program.

A specific aspect about the program I would change at a later date would be the lack of purpose the planet class has in itself other than being a store of data- with its sole purpose being only of returning its pointers to another class (sol_sys) without having a functionality of its own. This specific aspect about the program is I think very non-object oriented, as we seem to be letting a class take control of another class' private variables (in a containing relationship). To mend this situation, we can allow the planet class to create subsequent planets itself in a recursive fashion (like the recursive destructor) and therefore only allow the solar system to request adding/taking away/displaying another planet through its member functions. Additionally, another possible change to be made for expansion would be to generalize the classes further, and have a mass/position class as an ancestor to every other class that at construction randomly assigns a coordinate in 3D space and a movement vector/position of rotation which I can use with the time library to plot motion. Additionally, the spacecraft can then contain functions to accept galaxies and using the vector values received, adjust a flight path to the planet with life.

I did not need much use of any of the debuggers this program as the data structure assigned was relatively simple, with the pathways of execution being very limited- most of the functions were called at construction and the scope of errors were limited to easily rectifiable syntax errors. I did however use Valgrind to correct a few minor character array deallocation errors, which I could pinpoint due to its detailed line number provisions. On one occasion too, I resorted to using gdb to run through specific functions for incremental implementation, and put together working modules of the code very incrementally. The get_type() function was also initially a root cause of several errors right across the program, which I discovered later was due to un-freed memory and incorrectly allocated char pointers. Valgrind is I tool I found especially useful off late with large scale data structures as I could test several implementations of a particular structure and determine the most reliable form of implementation. Also a tactic I found that was time-saving was to note all the different line numbers pertaining to different

errors –memory or otherwise- and then make changes on the ones appearing too often before figuring out the rest. This allowed me to prioritize my attention depending on the ‘fatality’ of the error and therefore make continual progress.