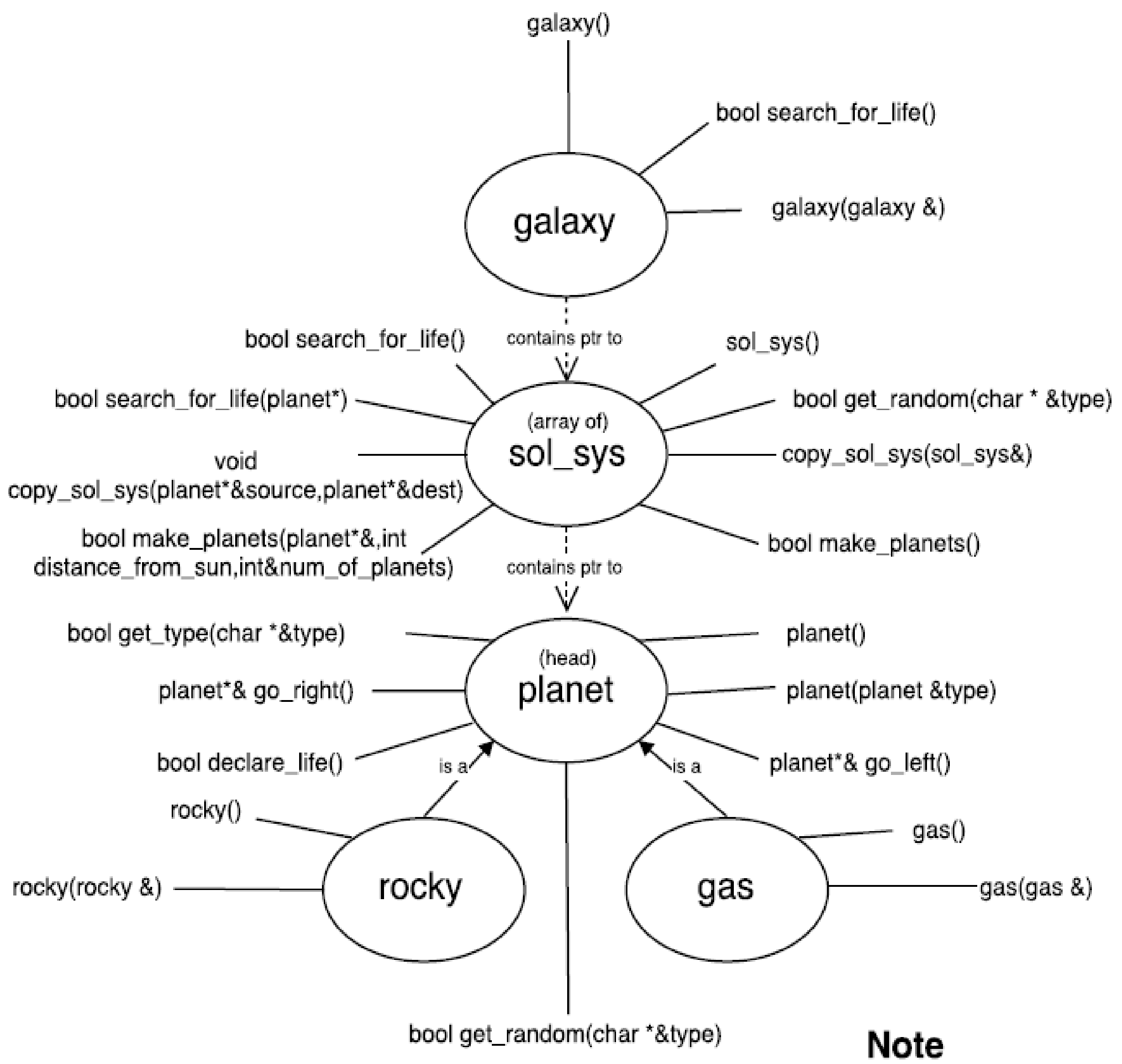
CS 202 DESIGN AND UML DIAGRAM FOR PROGRAM 1

BY TEJAS MENON; ASSIGNMENT BY PROFESSOR KARLA FANT



Note

A node class would not be necessary here as the planet itself would contain pointers to act as the node class. Each sol_sys would point to either a rocky or a gas but because they both inherit the planet class, a planet pointer can point to either one.

To simulate a galaxy, this program will include 5 classes, each with distinct purposes and operations. The all-encompassing galaxy class will contain a pointer to an array of solar systems, set to a random size and this value stored in a variable for later operations. Random number generation in this program will be achieved using the rand() function from the <cstdlib> library and numbers between any two values can be generated by rand() % range + lower_limit.

The default constructor of the galaxy will call the make_planets() function on each solar system which will then create a doubly linked list of planets. In essence, as soon as a galaxy object is created in the client program, a random number of solar systems will be generated each containing a random number of planets.

A few initial design considerations:

1) The default constructor of sol_sys will not include creation of the planets straight away as in the event a copy of a sol_sys is to be generated, the galaxy will still need to allocate memory to the sol_sys pointer first, prior to passing in the sol_sys to be copied. Therefore, this calls for a separate make_planets() function and a separate copy_sol_sys(sol_sys&) function, each of which will be called from the default and argument constructors of the galaxy class.
2) The planet nodes will contain next and previous pointers that can point to another planet, and therefore it's rocky or gas derivatives as well.
3) Functions and data members of the base class ‘planet’ will be those common to both subclasses ‘rocky’ and ‘gas’. In fact, both subclasses will only ‘affect’ the contents of the ‘planet’ variables such that its contents are unique to a rocky or gas planet. For example, a rocky planet will initialize lesser moons, a smaller mass/size value, no rings, more habitable atmosphere and pressure values to the ‘planet’s’ protected variables. Of course, these values will not be hardcoded but rather ‘weighted’ to allow for such a possibility. How this can be achieved is by tweaking the range and lower_limit of the possible outputs of rand() for each planet type, so that a large value for a mass or a highly unreasonable atmospheric pressure is less likely. For a more reasonable output, a random number generator that produces output on the basis of a normal distribution can also be chosen, but in this case avoided as this method will require more research and time to be provided to finding average planet sizes and mapping these to the random number generator.
4) Certain variables such as temperature, atmosphere type and velocity in the planet class will be dependent on more than one factor (whether the planet is rocky or gassy and also the distance of it from the sun). For such variables, the result of two ‘weighted’ rand() calculations will be averaged, and this value assigned to each respective variable.

   The planet class and it’s rocky and gas derivatives:

   Rather than using inner and outer as class types for a planet- as those keywords mean little about the actual composition and more about its distance from the sun- I used rocky and gas to determine the planet’s different characteristics and let the rand() function decide whether a rocky or gas planet is to be created next in the DLL. In particular, when the make_planets() wrapper function is called from the sol_sys class, this function can use rand() to set an initial sun radius/mass, the distance of the first planet from the sun and finally also the number of planets to be generated. Then this function can call the recursive make_planets() function passing in all the generated arguments which this function can use to:

1) ‘Weight’ the random generation to either a rocky or gas planet depending on the current distance from the sun.
2) Decrement the num_of_planets variable with each recursive call to finally exit the function when the value reaches zero.

Finally, the get_random() function for the planet and the sol_sys will use an initial ‘type’ or ‘noun’ entered to receive a random name for sun/planet/atmospheres from an external data file which will be stored back onto the type char*. This function will use rand() again to read from a particular line in the .txt file (the lower and upper limits set by the ‘type’).