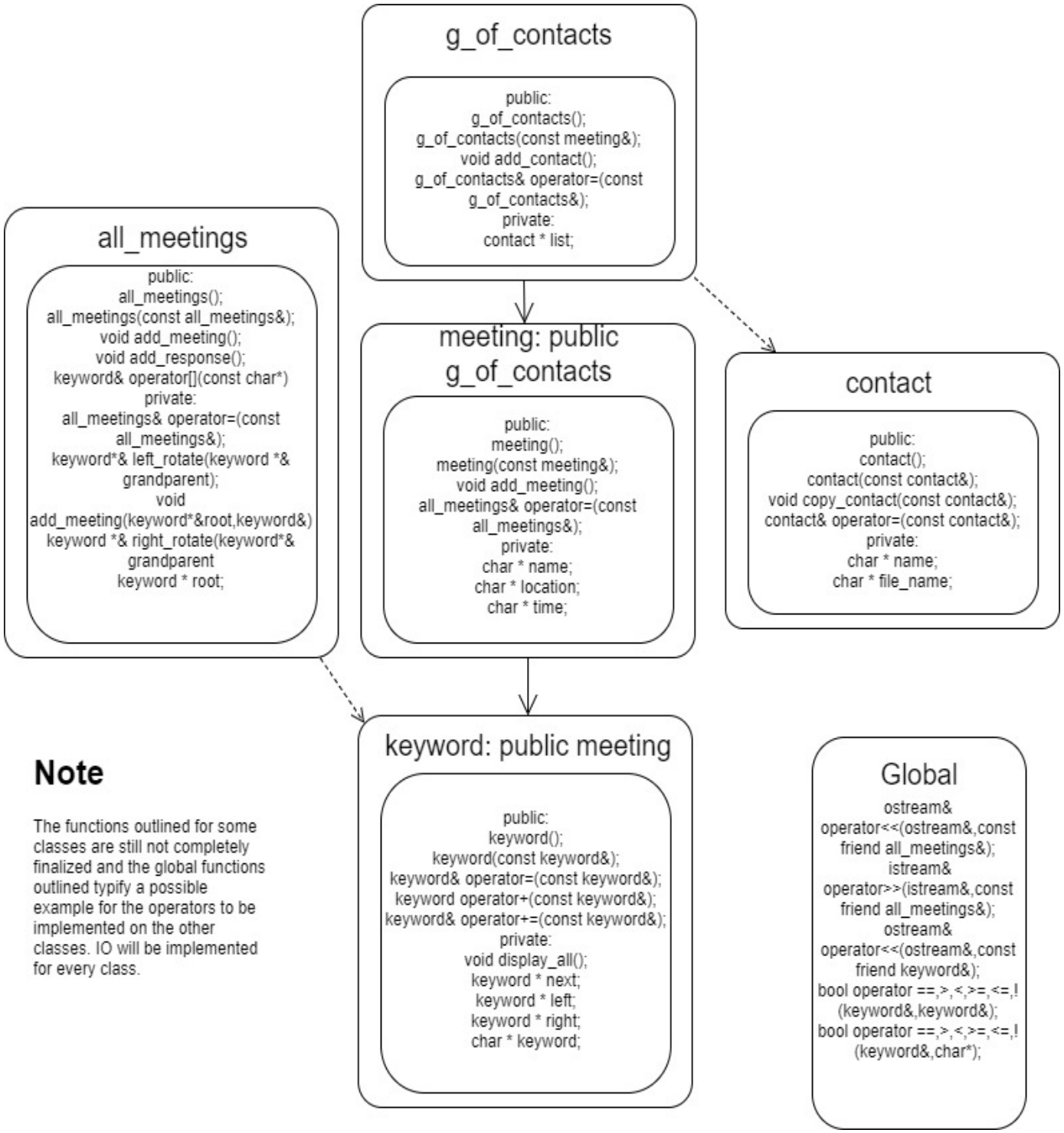
CS 202 DESIGN AND UML DIAGRAM FOR PROGRAM 3

BY TEJAS MENON; ASSIGNMENT BY PROFESSOR KARLA FANT



Note

The functions outlined for some classes are still not completely finalized and the global functions outlined typify a possible example for the operators to be implemented on the other classes. IO will be implemented for every class.

My event manager program consists of 5 classes, each managing their data and in some cases, even their respective data structures. By this standard, this program stands as unique since there isn't an umbrella data structure class totally managing all the nodes and connectivity. To provide this, my keyword class (that is a derivative of the meeting class, and an ancestor of the g_of_contacts class) not only form the nodes for the red-black tree structure, but also autonomously manages its linear linked list. In consequence, the keyword class would contain next pointers to other keyword objects (the meetings under the same keyword) and also left and right pointers to keyword objects of different keywords. The division of duty is such that the all_meetings structure constructs and manages the red-black tree while passing created keyword arguments to the keyword class functions which then add it to the beginning of the linear linked list. Therefore, the keyword class can support all linear linked list, arithmetic and relational operator functions to compare and add keywords and therefore not require external management from the all_meetings class. A recursive process similar to the recursive destructor that deleted next if it wasn't NULL can be utilized here, where searching/displaying/deleting can occur by one keyword object calling the same function recursively on the next keyword object and this process continuing until next is NULL or the correct meeting is found. This will allow for appropriate duty division between two data structure classes and maintain OOP standards- to each his own (objects don't share pointers between them).

The function of the g_of_contacts is to provide contacts their own personal responses under their own meeting and store each response in an external data file of a different name. What this implies is that the all_meetings class would contain a specific number that is incremented with each keyword (meeting) created and the external data file would be named uniquely using that number. This would allow for a contact to store responses multiple times in a particular file without the need for a ‘linear linked list of responses’ for example. At the end of a session, all these newly generated files can be removed using the remove() function from the <cstdio> library. Using an external data file this way would produce a response list for a particular meeting that recognizes the user that output a certain message and also be easier to manage than an LLL.

For inputting data from the user to parent objects particular class, I can use the scope resolution operator to call the >> function from the base class prior to requesting data from sub classes. This would allow for concise code that wouldn’t require the client performing I/O themselves and then inputting long argument lists. Additionally the same technique can also be used to check for equality using the relational operators and also in use of the assignation operator, wherein I can request base class assignations prior to performing one for the current class.

For displaying a particular LLL of meetings under a specified keyword, I can use the regular search algorithm for a binary tree and request the << function for each keyword that will then recurse through the LLL (untraditionally because I will not be traversing the LLL using from the ‘head’ keyword and rather will call the same function on *next (keyword object) until next reaches NULL. Searching would be an algorithm that would be performed in O(log n) due to the use of a balanced red black tree – and therefore any amount of keywords can be stored and accessed quickly.