Ryan Filgas
CS 163
Karla Fant
Program 1 Design Write Up - New Years Resolutions

My program will implement a class of new years resolutions entered by the user. The class will manage a linear linked list of resolutions, and each resolution will contain a sorted linear linked list of tasks. The user should be able to add and delete resolutions, add and delete tasks, and display resolutionsand their corresponding tasks. The class is an abstract data type, so as such it will only manage andoperate on the contained data structure. The data structure will be contained in the private section ofthe class, and will be masked from the user in all public function calls. Below are descriptions of each function followed by design considerations.

The public section of the 'Resolutions' class will contain:

add_new – add_new is a function to add a new resolution to the end of the list. The argument will be the name of a new years resolution (char array), and the return will be a boolean. The function will return true if a task will be added, false if the input is incorrect. The Main calling routine will output a print statement to the user after receiving a false return from this function, so the user and main function won't have to interact directly with the data.

delete_resolution – delete_resolution is a function to delete any resolution requested. The argument will be the name of the resolution the user wishes to remove, and the function will return True if there is a node to be deleted, and will delete the node. The user won't have to know what a node is, they'll just be deleting a resolution.

To do this we'll need to first test if there is a node, delete if there is only one, and if there are more, test each node for a name match. If it matches the previous pointer will point to currents next pointer, and the current pointer will be used to delete the node. If the node to delete is at the end when currents next node is null, the previous pointer will be set to Null, tail set to previous, and current will be deleted. If no match is found, the function will return false.

add_task - Is a function to let the user add tasks to a specific resolution. Arguments will be the name of the resolution (a char arrray), and a struct of type task.

The challenge in adding tasks is they need to be sorted. Since tasks are added one at a time, this function must insert in order. It will check each node to see if the current node is smaller, and if it is, the function will insert before this node using a previous pointer. The previous pointer will then point next to the new node, and the new nodes next will point to current. If there's an empty list, it will addto it, and if the insert is at the end, then tail's next pointer will go to the new node, and the new nodes next pointer will go to null. Then the tail pointer will be moved to point to the new node.

display_tasks - Is a function to let the calling routine display the tasks of a resolution to the user. It will take the resolution name as an argument and return false if the resolution name is invalid, or if there are no tasks to display. The user won't need to know about classes or structs, they'll just be answering questions, and therefore shielded from the inner data structure details.

The function will have to iterate through the LLL of resolutions checking each node until it doesn't find a match.

display_resolutions - is a function to display the full list of resolutions. No arguments are needed, and the function will display resolutions and return true if there are any resolutions to display. the user won't need to know about LLLs, or the data structure in order to use this function, they'll simply call it.

To do this, the function will need to iterate through the LLL displaying each resolution until the next pointer is null.

The private section of the "Resolutions' class will contain:

task_array - is an array of pointers to hold the tasks in a linear linked list. This will be the data portion of a resolution array.

resolution _array - is an array of pointers to hold each resolution in a LLL.

Structs to be called by private members of the resolutions class:

task_node - a struct containing the name of the task (a char array), how long the user plans to spend
(a float), the priority(1-10)(an int).

resolution_node - is a struct containing a resolution name (char array), a head and tail pointer to a LLL of task nodes, and a next pointer for the following resolution.

resolution_match - is a private function that takes in a function name, and the head of a list, and returns a pointer if there is a match, and false if there isn't.

Design considerations:

The main design considerations in this project will be adhering to the definition of an ADT, the data structure involved, efficiency, readability, memory management, and testing. These are considerations because the program needs to be error free, efficient, user firendly, programmer friendly, and well organized. It could be dealing with massive lists, and may need editing in the future by others.

•**ADT Structure** - This class will have to strictly follow the definition of an abstract data type. It must manage data, and operations on that data, nothing more, nothing less. This design hides the data structure from the user and the public functions only return true or false to indicate success or failure.

•**Data Structure** - The class will manage a list of resolutions, and each resolution will contain a sorted list of tasks. Inserting them sorted as they are added will be important, so that the program won't need to sort the list every time it displays.

•**Efficiency** - the program will use dynamically allocated arrays to use less memory, and functions should be optimized, for example the add_task function will insert before a lower priority task, rather than after, avoiding the inefficient alternative method of adding after nodes that are larger which would require extra conditions to work properly. For example there might be two 9s in a row, and an 8 needs to be inserted. The program would improperly insert the 8 between two 9s without added conditions.

•**Readability** - the program will follow design reccomendations, and simplify code by adding a node matchingfunction to search a list of resolutions for the desired resolution to edit or delete. As part of the design reccomendations, documentation in the comments, and proper naming convention will make the code easy to understand.

•**Memory Management** - In implementing the algorithm, memory management must be dealt with carefully. Properly implementing the constructor and destructors will be important. When closing up LLLs, stray pointers that aren’t temporary need to be set to null for memory management, so that other functions don't try to access the garbage they're pointing to.

•**Testing** - When searching through the resolutions list for name matches, the program must adjust for capital letters, and spaces, as user input may not be uniform. This is included in the test plan.