To begin with, I understood the difference between internal linkage and external linkage. Internal linkage means that any variable that is declared globally or locally can only be accessed within the module that they are declared in. To create an internal linkage/internal variable, we generally use the keyword static before the type of the variable to give them internal linkage. Also, static variables’ another property is that they can hold on to the memory allocated to them until the program ends, which means that data that was used to initialize them will still be available until the program ends. To give an example, in this workshop we created a local static variable inside the display function to print the count of daily routine of the user and incremented it at the end of the function. Every time the static variable was updated, next time the function was executed, the state was persisted in that local variable and that’s how we were able to print all the number of activities that a user performed throughout the day in an incrementing manner. Static variables have a good use case where we can create them as class variables to know how many objects are alive at any given moment since all the instances of that class share the same the class variable. We can increment them in constructors as objects are created and decrement them in destructor as each object gets deleted.

On the other hand, external linkage means that a variable is declared in one file, usually in header and is defined in another file, usually in implementation file. The external variables can be accessed throughout the project and there are no limitations as to which module can access these variables as far as they are declared & defined in the global scope. If they are declared and defined in a different scope, we must resolve their scope by either using the ‘using namespace identifier’ declaration or by prefixing it with namespace identifier followed by scope resolution operator. External variables are declared by using the ‘extern’ keyword followed by the type and identifier. Also, one thing to note is that using the extern keyword for the external variable, only declares the variable and for that reason it must also be defined, usually in implementation files. To give an example, in this workshop, we had to create a global external variable which was to be declared in header file, defined in implementation file and finally to be initialized and updated in the main module. From there we had to update the member variable based upon the value of the global external variable which was updated in main by looping over it. We were easily able to access the value of this variable since it was a global external variable, which are accessible across the whole project.

As far as upgrades are concerned, since we were dealing with dynamic memory allocation for a member variable, I added Copy Constructor, Copy Assignment Operator and Destructor, which is the default for any application that manages resources. I also created a private function that allocates the dynamic memory to localize the logic, since the same logic was used at least in 3 functions. And lastly, l added a function that returns me the time converted from seconds to HH:MM:SS format.