Name: Soham Thaker SID: 011-748-159 Email: [sdthaker1@myseenca.ca](mailto:sdthaker1@myseenca.ca)

To begin with, I learned 3 additional relationships that exist in C++ apart from Inheritance, namely, composition, aggregation & association. To understand them better, let us look at them individually to explain the concepts of these relationships.

Composition is a relationship that is more flexible than Inheritance. Classes that are in a Composition relation implement complete ownership of the members that exist within that class. Complete ownership means it is responsible for creating, copying, and destroying the resources that it owns, including the non-resource members. The members can themselves be copied by values, or by address(pointers/references) but they must be maintained by the Composer class itself. The class that manages the resources are called as Composer classes and the resources themselves are called as Component classes. The main function is not aware of this type of relationship. An example of Composition relationship from the workshop is Reservation and Restaurant classes. Both these classes manage the resources by deep copying the attributes and shallow copying the non-resources attributes in copy functions and move the object from the argument object into the current instance in move functions, hereby keeping a copy of attributes.

Aggregation is a relationship that is more flexible than Composition. Classes that are in an Aggregation relation implement partial ownership of the members that exist within that class. Partial ownership meaning it is not responsible for deep copying the resource attributes and destroying the resource or non-resource attributes. Which means for the copy and move functions we are only assigning the addresses that we receive from client into the attributes of Aggregator class meaning we share the identity(addresses)/resources in Aggregate class that client passes. This also introduces another concept for Aggregator classes where we can only have the attributes as a pointer or a reference since the resources will continue to exist outside of the Aggregator class. An example of an Aggregator class from the workshop is ConfirmationSender class which accepts a double pointer in a 2-argument constructor and the class itself implements a double pointer to the Reservation class. The double pointer is an array of pointers to object of type Reservation. In the move and copy functions we copy to the attribute, addresses of objects that the client passes to us, and client code manages(creation and destruction) the resources, for example ppReservations double pointer in main program. Also, it is preferred to for the Aggregator classes to make the attributes as constants so that we don’t end up changing the values held within the attributes since these objects are owned and managed by someone else.

Association is least coupled relationship between classes. It can also be referred to a service relationship. Classes in an Association relation exist independently of each other and are complete without the other classes. They do not involve any kind of ownership between classes. An example of Association would be the objects of iostream and iomanip libraries that we use regularly in C++. The classes that use the objects of iostream and iomanip can exist on their own if these objects are not implemented/used within the class members and classes do not own these objects either. If the dependencies do not exist, the classes in Association relationships will continue to exist.

Apart from that I learnt how to extract a token from a string and then perform some operations like trimming whitespaces off from the extracted token. This exercise helped me understand how to use string functions like find, substr, erase and stoi to perform operations on string objects.