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

Firstly, I learned how are classes defined. The two famous compound types: struct and class are used in C++ however, the latter one is widely used due to its various functionalities. Structures are public by default whereas, Classes are private by default and they consist of data and logic encapsulated altogether in one package. We can create as many instances as possible of a class. An instance/object of a class has access to any of its attributes and methods. It is created by writing the type of the object, which is the class name, followed by the object name. To instantiate an object in this workshop, we were to use setter method (there is a much better way than using setters, which is constructors(special functions), which are ran during the time of object creation and their sole purpose is to instantiate an object). Also, we can add keywords, private and public any number of times in a class and we decide which data members and member functions needs to be private or public. If a member function/data member is private, then client code has no access to these variables/functions since they are private and only the member functions can access them. Also, when writing the definitions of member functions, it needs class name followed by the scope resolution operator, followed by the function name and parameters if any. The methods of a class can be of 3 types, query, modifiers & special. The query functions which cannot change the state of the object, they can only report the state of the object, they are usually referred to as getters whereas modifies are free to change the state of the object and are referred to as setters. If we are to access a member function or a data member, it can be called/accessed as object followed by the identifier of method or identifier of the instance variables.

Secondly, I learnt what safe empty state and data validation is and their benefits. When we have a class and its data members are declared as private, client code cannot access those variables. To set the instance variables with a client provided value, we use setters(which are public and can be accessed by the client code), where the data validation occurs. When a member function is called, we check whether any of the explicit parameters have a value that is impossible for a specific data member, if it is then we set the instance variables of that object as an empty state, which would be some type of a value that is not possible for a variable to contain. Otherwise, we accept the data that is being passed to the setter and set the instance variables with that data. After implementing safe empty state and data validation functionalities to our code, we can make our code more meaningful if we were to display the values of any given attributes. If the object is in empty state, that means it does not contain any data, and we can simply print on the console, ‘no data available’, whereas if there is any valid, we print out the valid data stored in each instance.

Thirdly, I learnt formatting conversions and how to format data inserted into the output stream. There are built-in methods written in ostream class for output stream objects which is responsible for handling output stream. I used width, fill, setf, unsetf and precision in this workshop to fulfill the output requirements. The cout.width(5) method sets the field width to the integer value passed as a parameter. And usually cout.fill(‘\*’) method goes hand in hand where the cout object prints any data that is to be printed on output stream and cout.fill(‘\*’) pads the rest of the width with a char ‘\*’. Important to note is that width settings only apply to the next data that is to be outputted, whereas fill settings apply to all the following data in the output stream. Besides, cout.setf(ios::left) help in setting the alignment of data that is to be printed on the screen, which means we can set left or right flag and it will print the data according to the flag that is passed. In order to unset the alignment settings, we use cout.unsetf(ios::left) to align the output data back to the default settings. Whereas cout.precision(2) helps in printing a double or float number to the specified precision. Usually, the default format for the floating-point numbers is general format which goes by the number of significant digits and we might not get the result as expected in cout.precision(2). In order to print the number with exactly 2 decimal points, we can use cout.setf(ios::fixed) where ‘fixed’ flag counts the number of digits after the decimal point and we can get our expected result. Later on if we want to apply the general format to the following numbers, we can unset the flag by writing, cout.unsetf(ios::fixed).