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

In this function I learnt what are templates and their uses. Templates functions are used as generic functions in generic programming that can accept any types, be it primitive or custom as their template arguments. If a function performs the same functionality across multiple types, a much better option as a programmer is to use template functions that accept any type of parameters passed to it and performs some logic written within that function. Template functions are compiled at compile time which is why they must be implemented in the header so that they can be called from the source files when we include the header where they are defined and declared. The syntax of a template function looks like, “template<typename T>” and the subsequent code of block is identified as templated function. Here in the syntax, typename/class is used as an identifier of the type which the template function will support, and ‘T’ is replaced by the type passed as template argument at the time of calling the function. To call a template function whose function prototype is, “template<typename T>; void add(T arr[], int size) {some logic here}” that will use the type as int, we call that function from the client code like this, “add<int>(arr, size);” here the template arguments “<int>” identify what will be the type of ‘arr’ and the function will execute the logic written within the add function. The explicit identification of arr’s type which is an ‘int’ when calling the template function is not always needed since compiler can sometimes figure out by itself the type of template arguments like in this case which is a primitive type and wont be needed to explicitly tell the compiler the type of arr. Besides, if we want to identify arr as a custom type where the logic to be performed is to be the same, then we can call the add function as “add<Container>(arr, size)” and now the template will recognize the arr’s type as a custom type, Container. However, if in the function we are using any of the operators provided to us by the language to perform some arithmetic or logical operation, lets say compare or assign or negate or use any operator to achieve the end result, we must make sure to overload these operators as part of the custom types’ class definition since the language does not provide overloaded operators for custom types like it does for the primitive types and hence it is on us as a programmer to write the code that overloads the operator that we use in any of our template functions. For example, in the workshop, ‘search’ function must have == operator overload for any custom types passed as the template argument to compare if the string is present in any of the data members of the custom type and same goes with ‘listArrayElements’ function which uses the << operator where we must overload the insertion operator in any custom types used in this function to print the data members of any custom types passed into the template function.

Apart from that, I also came across the class templates and their uses. They are very much similar in terms of the usage as compared the function templates with only one difference which is class templates’ object at the time of creation support a specific type which they will represent throughout their lifetime and we can preform similar logic defined in the class templates for any type of the object created for the class template be it primitive or custom. Also, all the functions in a class template are a template function so at the time of writing their definition we must identify them as a template function like so, “template<typename T>”. The template arguments can be added but this is the general format. Also, the template arguments must always be known at compile time be it class templates or function templates otherwise it can create compilation errors.