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

Firstly, I learnt how to work with binary files. Binary files are not so user friendly because a user cannot read what is stored as its content when opened on a desktop software, since the content itself is 0’s and 1’s. Binary files provide fast access to the contents stored on it when reading or when writing data to the file. Another benefit that it provides is that the content that we store on a binary file would stay exactly like it was once we read data from the file. Since we perform encoding of the data and then store that encoded data to a binary file, we are provided an assurance by using binary files that no data would be lost once we read the stored data from the file.

Secondly, I learnt the uses of std::bind. std::bind is a variadic function that accepts any number of arguments and returns a functor which we can call at any point. We store the returned value into a local variable and then we only need to invoke the local variable by using the () which will perform a call to the overloaded compiler generated function object’s () operator. This operator will in turn call the function that we passed as its first argument and passing to it the bound arguments. So, it basically wraps around the function that we pass as its first argument, later to be invoked as regular function object, without passing any arguments since the arguments were already bound the function at the time of the call to bind function. Also std::bind provides a neat functionality where if we don’t know the placeholders right away, we can leave that up to the code/function that will be executing the bound function. All we write is std::placeholders::\_n where n is a number starting from 1 up to an implementation defined maximum number. This placeholder will be replaced with something which the code/function provides that calls the bounded function. Binding a function to its arguments is important since later when we want to call the function the arguments are already bound to that function object, all we must do is make a call to overloaded () operator of the returned functor. Also std::bind was helpful in this workshop since if we want to pass more or less arguments to the converter function we can do so just by simply adding extra arguments or removing arguments. Since bind is a variadic templated function it can accept as many arguments as possible and can return us a complete wrapper for a function object that we can later make a call to the function itself which is wrapped underneath the returned function object.

Thirdly, I got to work with multithreaded application. The advantages of a multithreaded application are huge. They provide parallel execution for an operation that might take longer to execute when executed on a single thread. Multithreading works by distributing tasks across hardware threads that perform a task by working on chunks of data. Thread object’s constructor accepts a function which it can call to perform a parallel execution on a task. In this workshop we were supposed to create 4 thread objects, assign them a task to complete which was properly partitioned and distributed equally amongst all 4 thread objects. The threads performed the encoding/decoding in chunks of data, which was done parallelly, among 4 threads which makes the application faster since the main task of this application which was encode/decode was distributed among 4 threads. Once the threads were done executing, we synchronized the threads so that no undefined behaviour happens. This specific application was not so good example that would’ve benefitted from parallel execution since the task to perform was trivial.