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

The project taught me one key concept about how to dynamically allocate create a C style string array where the size of the incoming string from user is unknown without using the string class itself. The procedure for handling an unknown string size and then allocating it to a dynamic array is as follows. A static array of a fixed size is created. And we use a loop as far as the istream object is good. Then according to the stream, whether an input is coming from a file or from the console we use istream object’s get method to get the input and store it into the static array. If the input is coming from the file, we check for the eof as a delimiter otherwise we usually check for the new line when the input comes from the user, however that delimiter can be any other character that the programmer wants to check for when to mark the end of the input. Once the data is stored into the static array we then check if the istream object was good or it failed. If it failed then no need to proceed further, end the loop and clear the state of the istream object. However, if the object is good, there are a bunch of steps that we need to perform to successfully extract all the characters from the input buffer and store them onto the dynamically allocated array. First, we find the length of the static array so that we can keep on increasing the size of the temporary array pointer and eventually the original array pointer as well. After this, a temporary pointer is allocated as a char array with a size of the static array’s size + the original array pointer’s size + 1 for the null byte. Then, we do a strncpy from the original array to the temporary array until the size of the original array. Once done, we again do the strncpy from the static array to the temporary array’s element which would start from the size of the original array until the size of the static array. The we mark the last element of the temporary array as null byte. Once all of this is done, we delete the original array pointer so that we can assign the temporary array pointer’s address to the original array pointer. Temp and original array pointer point to the same address now so temp array does not need to be deleted, deleting temp array would result in crash since original does not point to that address anymore since it was freed and if we try to print original array’s values it does not exist because that address does not exist. Lastly, we assign temp a nullptr value so that if it were to be used again, it would be safe to do so. We must also increase the size variable that is the original array pointer’s size with size += size of the static array. If there is more user input, we continue the same process again until the delimiter is not reached if the input is coming from the console. The way to do that would be using the delimiter which is a new line usually. Since we used get method to get the user input, the delimiter is still in the input buffer for the programmer to clear it and we can check if the length of the static array that extracts the input and stores the data into itself is less than its size then we know the delimiter is reached and thus the input has ended so its time to exit the loop. We clear the input buffer that contains the delimiter and exit the loop. We can then return the original array pointer address as a return value for the function and any other function can use that value throughout the program, but we must delete that memory at some point when we do not need to use it since the memory was never freed in the function where it was allocated.

Apart from that, I have been localizing the logic a lot throughout the project. This helped me avoid code duplication. Wherever I saw the opportunity to localize the logic I created a function that contains same logic so that I can call the overloaded function from other functions that use the same logic.

As far as challenges are concerned, anything that is new to us feels always challenging but over time as we practise, we get better at it. For me all these concepts were tough since they were all new to me, but I eventually figured them out as I understood them thoroughly.

As far as feedback for this course is concerned, there is no feedback from me. My overall experience learning C++ was pleasant and its concepts related to Object Oriented Paradigm were great. I loved doing workshops and project. They both were the best combination for me to practise coding for C++ and they were insightful and helpful in understanding C++’s OOP concepts which helped me do well in this course.