5-2 Activity: Encryption Coding

**Screenshot**

**A computer screen with white text

Description automatically generated**

**Summary**

As a proof of concept for a manager that wants to incorporate encryption into a banking application, we were tasked with demonstrating how XOR-based encryption would work with some partially constructed code that we were tasked with finishing. This assignment involved the implementation of several key functionalities such as XOR-based encryption, loading data from a text file into a string, and saving a string to a text file. Completing this assignment involved implementing several TODO comments within the encrypt\_decrypt, read\_file, and save\_data\_file functions.

Starting with the encrypt\_decrypt function, this function takes a string and key as parameters and performs encryption or decryption on the passed string using the key. Within this function, it was required to implement the XOR encryption or decryption in the for loop that iterates character by character through the passed string. This was accomplished by adding the line, output[i] = source[i] ^ key[i % key\_length];. This line essentially performs XOR encryption or decryption on each character in the source string that corresponds to the character in the key. The % modulus is utilized so that the key repeats if the key is shorter than the length of the string passed.

Moving on to the read\_file function, this function is responsible for loading a file into a string. The TODO requirements within this function were to implement the logic of opening the file and appending the file to the file\_text string to be returned by the function. The approach taken for this requirement was to use ifstream to open and read the data from the file with the line, std::ifstream file(filename);. The logic of the function is then encapsulated in an if-else statement. That is, if the file was open then the file was read into a buffer with the line, buffer << file.rdbuf();. The content of the buffer is then appended to the file\_text string with the line, file\_text += buffer.str();, and then closed. If the file could not be open for any reason, then an error message is printed that says, “Unable to open file.”

The final function that required implementation of TODO comments was the save\_data\_file function. This function was responsible for saving file data in a very specific format provided by the comments. Line 1 required printing the student\_name variable, line 2 required printing a timestamp, line 3 required printing the key used, and line 4 required printing the file data. Output file stream was utilized to open the file with the line, std::ofstream file(filename);. A simple if statement is then used that checks if the file open and prints an error message if the file is not open that states, “Unable to open file.” Printing the student\_name variable for the first line was achieved with the line, file << student\_name << std::endl;. The second line was a bit more complex and required implementing logic for getting local time and formatting the local time in the requested format of “yyyy-mm-dd.”

The approach taken involved first creating an array that stores the timestamp with the line char timestamp[80];. A struct is then created to hold the time information in the line, struct tm time\_info;. Next, local time is retrieved with the line, time\_t current\_time = time(0);. Current time is then convered to local time using localtime\_s in the line, localtime\_s(&time\_info, &current\_time);. Then, local time is formatted and stored in the created timestamp array with the line, std::strftime(timestamp, 80, “%Y-%m-%d”, &time\_info);. This allows for printing the timestamp for the line 2 requirements with the line, file << timestamp << std::endl;. Line 3 and 4 were easily addressed with the lines, file << key << std::endl for printing the key, and file << data; for printing the data.