**CS 405 Module Five Encryption**

For this assignment, the goal was to implement XOR encryption, read from a text file, encrypt the data, save it to a file, decrypt the encrypted data, and finally save the decrypted data to another file. Several adjustments were made to the code to fix errors related to file paths, encryption logic, and file handling.

**Using Relative Paths**

I made the program to use **relative paths**. A relative path is a file path that starts from the directory where the program's executable is running, instead of specifying the full path from the root directory of the computer's file system.

By using relative paths:

* **Benefit**: The program will now look for the input and output files in the same directory as the program's executable file, regardless of what computer it's run on.
* **How It Works**: When someone unzips the project folder, as long as the text files (e.g., inputdatafile.txt, encrypteddatafile.txt, and decrypteddatafile.txt) are included in the same folder as the executable, the program will be able to find and use them correctly.

// Using relative file paths

const std::string file\_name = "inputdatafile.txt";

const std::string encrypted\_file\_name = "encrypteddatafile.txt";

const std::string decrypted\_file\_name = "decrypteddatafile.txt";

**2. XOR Encryption Logic**

The encryption method used in the assignment is XOR-based encryption. This is where each character in the input string is XORed with the corresponding character from the key. If the key is shorter than the input string, the key will loop around using the modulus operator (%) to ensure all characters are encrypted. I fixed the encryption logic by making sure it correctly handled different key lengths.

Here is the updated code for the encryption logic:

output[i] = source[i] ^ key[i % key\_length];

* In this case, source[i] is the character from the input string, and key[i % key\_length] picks the correct character from the key, wrapping around when necessary.

This change ensures that all the characters are encrypted and decrypted properly, and this is proven by the result shown in Screenshot 2, which displays the decrypted file decrypteddatafile.txt. As you can see, the decrypted file content matches the original content from the input file, meaning the encryption and decryption logic worked perfectly.

**3. File Handling and Saving Data**

Another part of the assignment involved saving both the encrypted and decrypted data to separate text files. This had to be done in a specific format, which includes the student’s name, the current date, the key used for encryption, and the encrypted/decrypted data.

To handle the date correctly, I used the localtime\_s function, which is the safer version of localtime. This prevents any security issues that could arise from using an outdated method. Here's how the date is handled:

time\_t now = time(0);

tm ltm = {};

localtime\_s(&ltm, &now);

std::stringstream date\_stream;

date\_stream << 1900 + ltm.tm\_year << "-" << 1 + ltm.tm\_mon << "-" << ltm.tm\_mday;

In this code, I used the tm structure to get the current date and formatted it into a string. This string was then written to the output files along with the student's name, key, and data.

* The encrypted file encrypteddatafile.txt, where the data has been successfully encrypted and saved. The content of this file looks like random characters, which is exactly what we expect from an encrypted file using XOR.
* The decrypted file, proving that after decryption, the data returns to its original form.

4. Final Program Execution

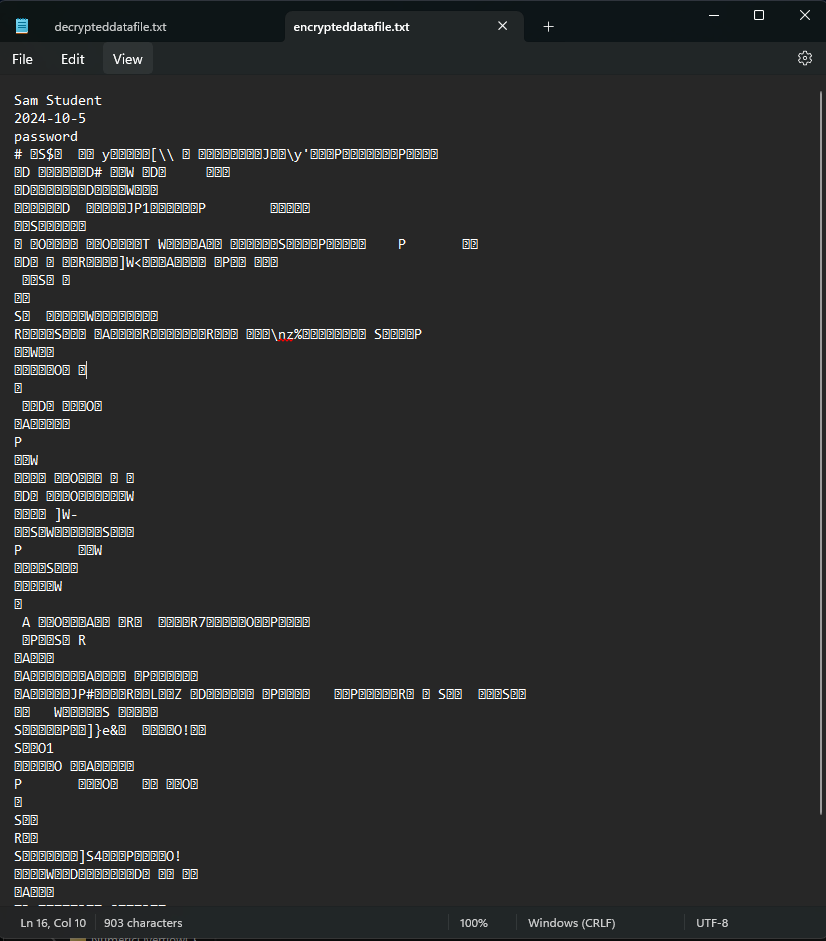
The final step involved running the program to confirm everything worked smoothly from start to finish. Screenshot 4 shows the console output where the program successfully read the input file, encrypted it, and then decrypted it. The output shows the paths of the input, encrypted, and decrypted files, indicating that the program executed without any errors.

This successful execution was achieved by making sure the following components worked together properly:

* Correct file path handling
* Proper XOR encryption and decryption logic
* Safe date handling with localtime\_s
* Correct reading and saving of text file

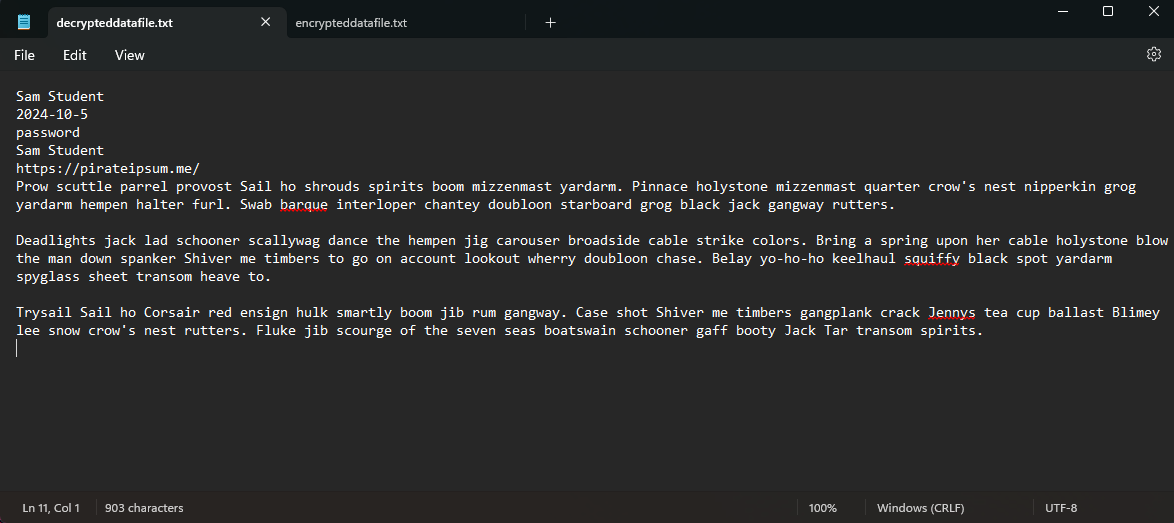
**Encrypted Data File (encrypteddatafile.txt)**

* Explanation: This file shows the encrypted data that was produced using XOR encryption on the original input file. The encryption process took the source text and the key ("password") to produce a file of garbled symbols, which indicates the encryption was successful. The key reason for using encryption is to ensure that sensitive data, like personal details, is unreadable if intercepted. This file represents how the XOR encryption transforms the original data into an unreadable format using the provided key.
* Placement: This should be shown as evidence that the encryption process was executed successfully. You can mention this when discussing how the encryption process works and how the file is rendered unreadable.



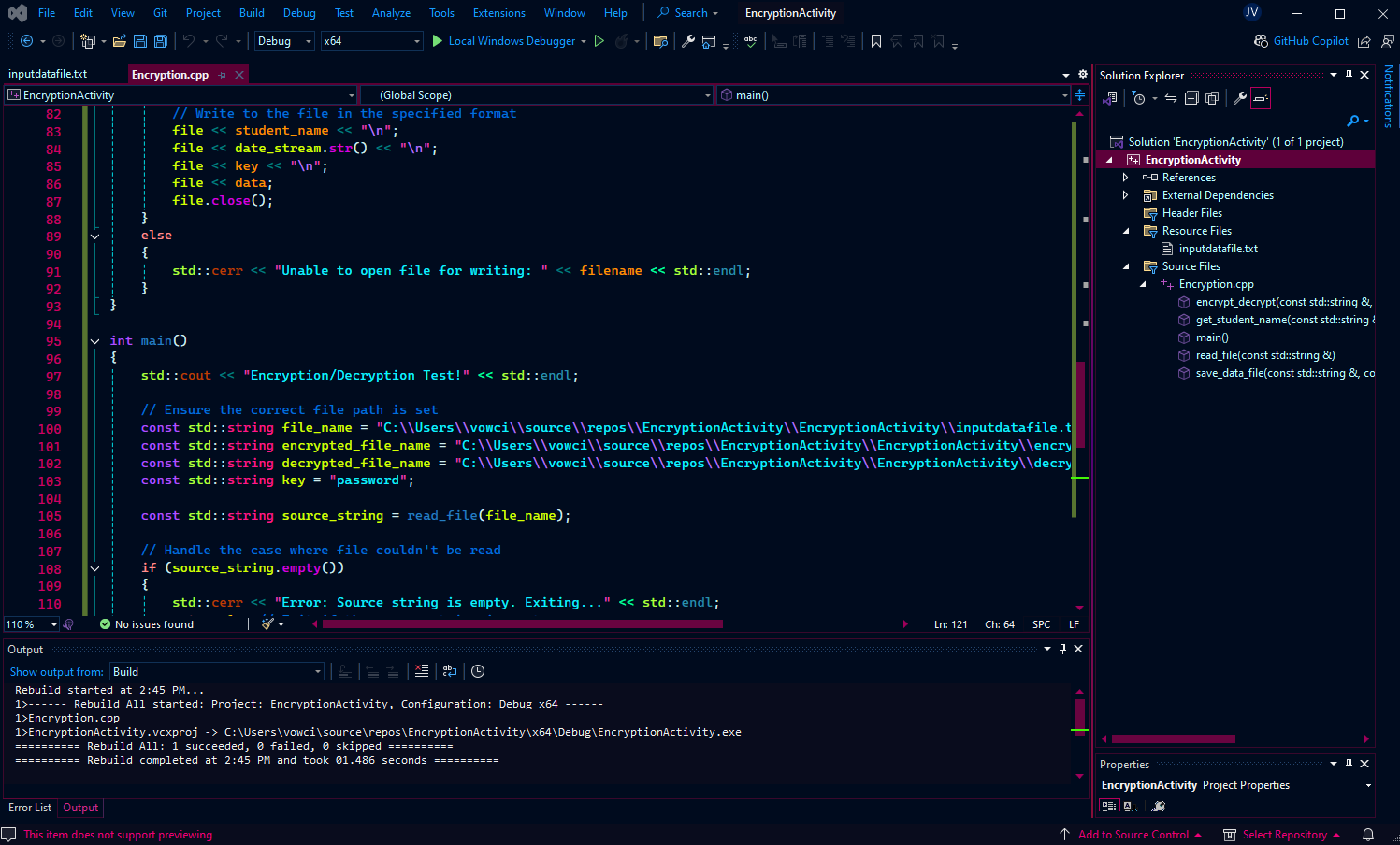
**Decrypted Data File (decrypteddatafile.txt)**

* Explanation: This file shows the decrypted data after applying XOR decryption using the same key ("password") to the encrypted file. As expected, the data matches the original input, confirming that the encryption and decryption processes were successful. This step is crucial to demonstrate that while data is encrypted for security purposes, it can still be retrieved in its original form with the correct key.
* Placement: Include this when explaining the decryption process and how the encryption algorithm can successfully revert the encrypted data to its original state.



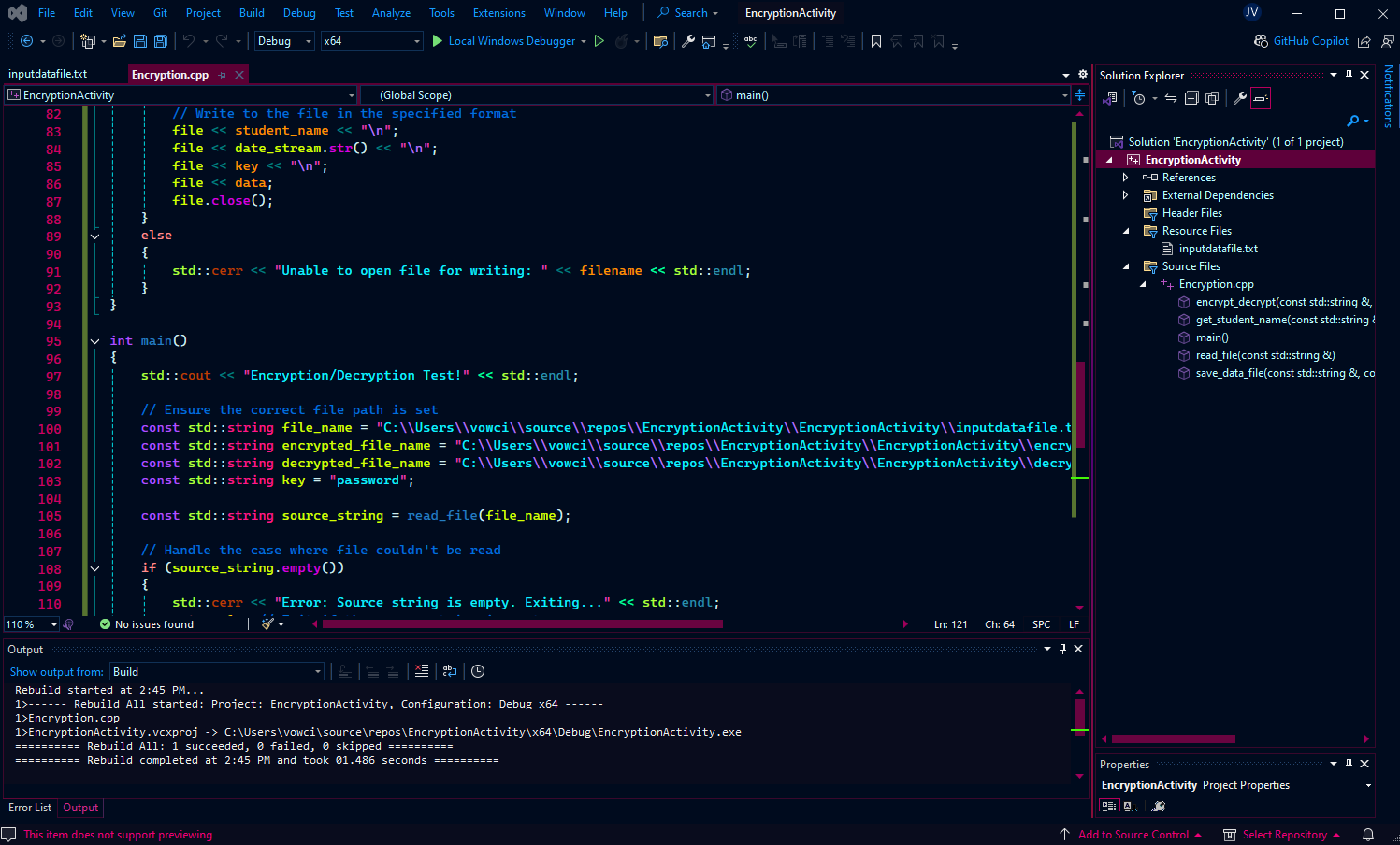
**Visual Studio Output Console**

* Explanation: This is the output from running the program, showing that the process of reading the input file, encrypting it, saving the encrypted file, decrypting it, and saving the decrypted file has completed successfully. The paths to the input, encrypted, and decrypted files are displayed, which confirms that the program handled the file operations as expected.
* Placement: Use this when discussing the overall success of the program execution. It provides evidence that the program functioned correctly without errors.



**Updated C++ Code in Visual Studio**

* Explanation: This shows the final version of the C++ code. The paths for the input, encrypted, and decrypted files are specified using the full file path. The read\_file, save\_data\_file, and encrypt\_decrypt methods are all visible and functional. This is the core code that processes the encryption and decryption tasks, demonstrating how the XOR algorithm works with file input/output operations.
* Placement: This is important when explaining the modifications to the code and how it was made to handle the file operations. You can walk through this image to describe the logic and functions that make the program work.



**References:**

ISO/IEC 14882:2017 (2017). *Programming languages — C++*. International Organization for Standardization.

Microsoft Developer Documentation. (n.d.). *File handling in C++ (fstream)*. Retrieved from <https://learn.microsoft.com/en-us/cpp/standard-library/fstream?view=msvc-170>

Seacord, R. C. (2013). *Secure coding in C and C++* (2nd ed.). Pearson Education.