Project 01

Due: Friday, March 3rd 11:59pm

1 Description

The project will be implemented as three separate programs. There will be a logger – responsible for logging all activity. There will be an encryption program – responsible for encrypting and decrypting strings. There will be a driver program that will interact with the user to use the encryption program. The entire system will be ran by running the driver program, which will launch the other programs and communicate with them through pipes. If you use C/C++ to code your project you must use the linux system calls fork,pipe, and dup2. If use Java to code your project you must use the Process class. If you use Python use the Subprocess module. Examples for all these approaches will be provided in class. Details of each of the programs are below.

2 Details

2.1 Logger

The logger will write log messages to a log file. The log messages are lines of text where the first sequence of non-whitespace characters is considered the action, and the rest of the line is considered the message. The log message will be recorded, with a time stamp in 24 hour notation, in the log file as a single line using the following format:

```
YYYY-MM-DD HH:MM [ACTION] MESSAGE
```

So, the log message "START Logging Started." logged March 2nd, 2022 at 11:32 am would be recorded as:

```
2022-03-02 11:32 [START] Logging Started.
```

The logger program should accept a single commandline argument – the name of the log file. On start, the logger program will open the log file for append (the current contents are not erased), and log "START Logging Started." The logger program will then accept log messages via standard input until it receives "QUIT". Before exiting, the logger program should log "STOP Logging Stopped."

2.2 Encryption Program

The encryption program should accept *commands* given as lines via standard input. The first word (sequence of non-whitespace characters) should be treated as a command, and the rest of the line(after the first space) as the argument for that command. Output is printed to standard out as a line of text where the first word is a *response type*. The currently set key is remembered. The encryption program should handle the following commands:

PASSKEY – Sets the current passkey to use when encrypting or decrypting.

ENCRYPT – Using a Vigenère cypher with the current passkey, encrypt the argument and output the result. If no passkey is set output an error.

DECRYPT – Using a Vigenère cypher with the current passkey, decrypt the argument and output the result. If no passkey is set output an error.

QUIT – Exit the program.

The encryption program has the following response types:

RESULT – The preceding command succeeded. The rest of the line (after a space) is the result of the executed command.

ERROR – The preceding command failed. The rest of the line (after a space) is the error message.

More information on the Vigenère cypher can be found at

https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher.

2.2.1 Example Interaction

Below is a single run of the encryption program.

- Input: ENCRYPT HELLO
  Output: ERROR Password not set
- Input: PASSKEY HELLO
  Output: RESULT
- Input: ENCRYPT HELLO
  Output: RESULT OIWWC

2.3 Driver Program

The driver program should accept a single commandline argument – the name of the log file. Upon start, the driver program will create two new processes, executing the logger and the encryption program. Pipes should be used to connect to their standard input and standard output. Python and Java provide streams to communicate over these pipes. C/C++ will need to use the read and write system calls to communicate.

Once set up, the driver program should prompt the user for commands, looping until the quit command is received. Each command should be logged, and the result of each command should also be logged. The start and exit of the driver program should be logged. All strings entered to be encrypted or decrypted should be saved in a history that lasts only for this run. The driver program should accept the following commands:

password – Provide the user with the option of using a string in the history or entering a new string. If a new string will be used, prompt the user for a password, and then set it as the current password in the encryption program. If the history will be used, provide the user with a menu where a number can be used to select a string stored in the history. The entered password is not stored in the history.

encrypt – Provide the user with the option of using a string in the history or entering a new string. If a new string will be used, prompt the user for a string, record it in the history, and send an encrypt command to the encryption program. If the history will be used, provide the user with a menu where a number can be used to select a string stored in the history. The results should be printed to standard output and saved in the history.

decrypt – Provide the user with the option of using a string in the history or entering a new string. If a new string will be used, prompt the user for a string, record it in the history, and send an decrypt command to the encryption program. If the history will be used, provide the user with a menu where a number can be used to select a string stored in the history. The results should be printed to standard output and saved in the history.

history – Show the history.

quit – Send QUIT to the encryption program and logger, and then exit the program.

In the above commands whenever the history is used, provide the user with a means to exit the history and enter a new string.

3 What to Turn in

Upload your submission as a zip archive containing the following:

- Source code (c, c++, java, or python files)
  - Source code should not require a particular IDE to compile and run.
  - Should work on the cs1 and cs2 machines
- Readme (Plain text document)
  - List the files included in the archive and their purpose
  - Explain how to compile and run your project
  - Include any other notes that the TA may need
- Write-up (Microsoft Word or pdf format)
  - How did you approach the project?
  - How did you organize the project? Why?
  - What problems did you encounter?

  - How did you fix them?
  - What did you learn doing this project?
  - If you did not complete some feature of the project, why not?
    * What unsolvable problems did you encounter?
    * How did you try to solve the problems?
    * Where do you think the solution might lay?
      · What would you do to try and solve the problem if you had more time?

4 Grading

The grade for this project will be out of 100, and broken down as follows:

Followed Specifications	50
Use of Processes and Pipes	20
Correct Output (User interaction, Menu, Log file)	20
Write-up	10

If you were not able to complete some part of the program discussing the problem and potential solutions in the write-up will reduce the points deducted for it. For example, suppose there is a bug in your code that sometimes allows two customers to approach the same worker, and could not figure out the problem before the due date. You can write 2-3 paragraphs in the write-up to discuss this issue. Identify the error and discuss what you have done to try to fix it/find the problem point, and discuss how you would proceed if you had more time. Overall, inform me and the TA that you know the problem exists and you seriously spend time trying to fix the problem. Normally you may lose 5 points (since it is a rare error) but with the write-up you only lose 2. These points can make a large difference if the problem is affecting a larger portion of the program.