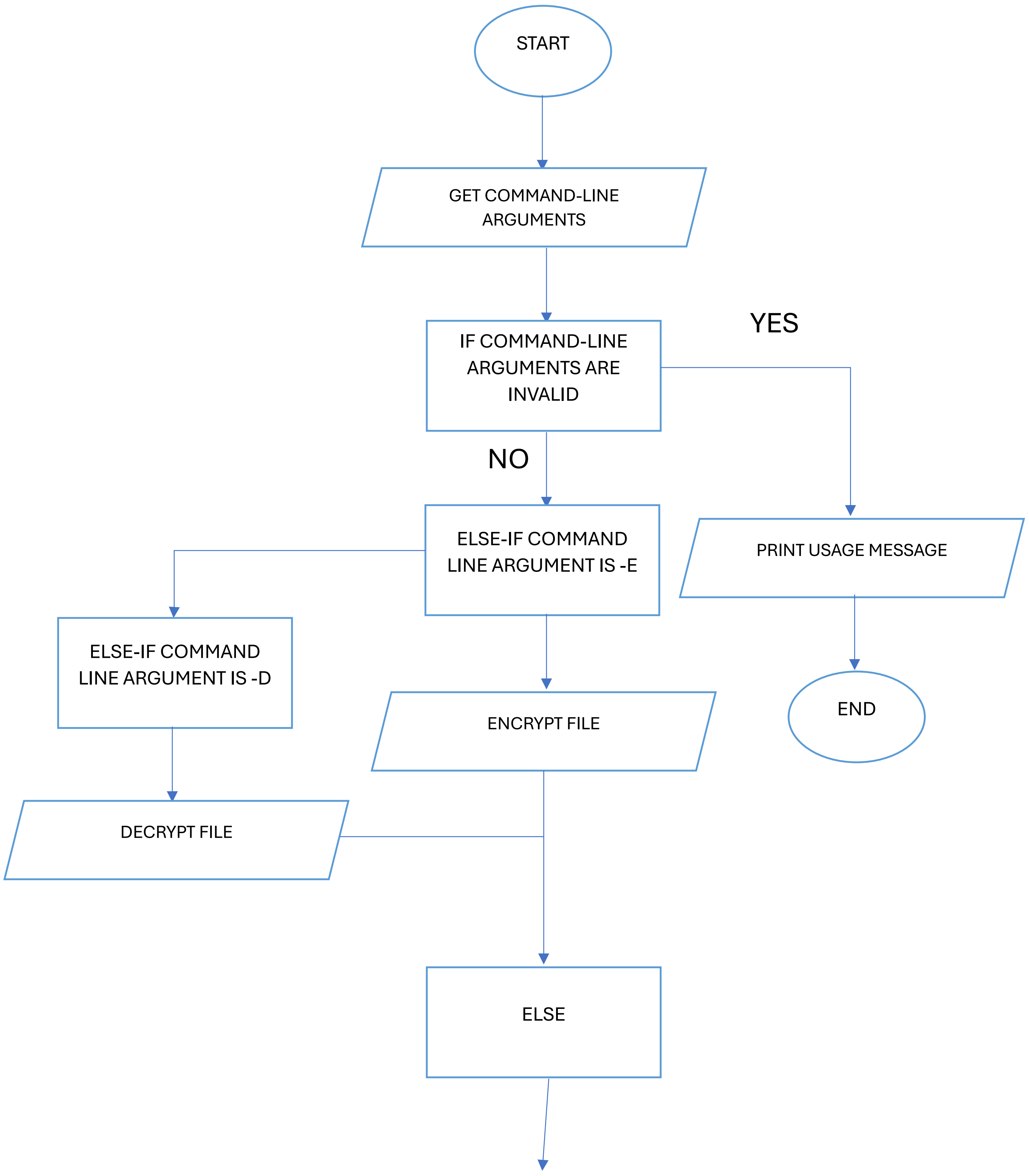
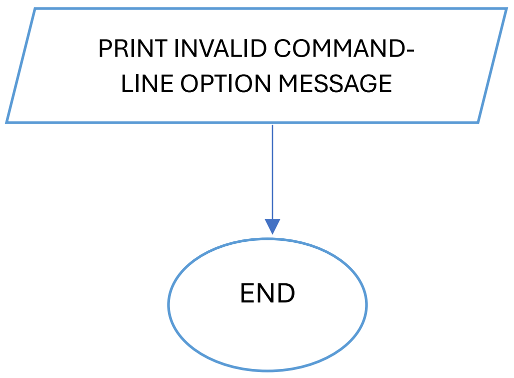
ENGG*1410: Introductory Programming for Engineers
Mini Project #1: “Encryption/Decryption Utility”

Mann Modi
Date: November 2, 2023

Preparation File

1. Flowchart


START
GET COMMAND-LINE ARGUMENTS
IF COMMAND-LINE ARGUMENTS ARE INVALID
YES
PRINT USAGE MESSAGE
END
NO
ELSE-IF COMMAND LINE ARGUMENT IS -E
ELSE-IF COMMAND LINE ARGUMENT IS -D
ENCRYPT FILE
DECRYPT FILE
ELSE



2. Pseudo Code

```
Function Encrypt(filename):
    hexBuffer = Allocate memory for a 3-character buffer
    open inputFile as read binary
    if inputFile is null:
        Display an error message
        Return

    open encryptedFile as write
    if encryptedFile is null:
        Display an error message
        Close inputFile
        Return

    Loop until end of inputFile:
        Read a character ch from inputFile
        Get the ASCII value of ch
        if ch is a newline character:
            Write "TT" to encryptedFile
        else:
            Calculate outChar = (ASCII - 16)
            if outChar < 32:
                outChar = (outChar - 32) + 144
            Convert outChar to a hexadecimal string
            Write the hexadecimal string to encryptedFile

    Close inputFile
    Close encryptedFile
    Free hexBuffer

Function Decrypt(filename):
    hexBuffer = Allocate memory for a 2-character buffer
    open encryptedFile as read
    if encryptedFile is null:
        Display an error message
        Return
```

```
Determine the output file name
open outputFile as write
if outputFile is null:
    Display an error message
    Close encryptedFile
    Return

Loop until end of encryptedFile:
    Read two characters c1 and c2 from encryptedFile
    Concatenate c1 and c2 to hexBuffer
    if hexBuffer is "TT":
        Write a newline character to outputFile
    else:
        Convert hexBuffer to an integer outChar
        outChar = outChar + 16
        if outChar > 127:
            outChar = (outChar - 144) + 32
        Write outChar to outputFile

Close encryptedFile
Close outputFile
Free hexBuffer
```

ENGG*1410: Introductory Programming for Engineers
Mini Project #1: “Encryption/Decryption Utility”

Group #15: Mann Modi
Date: November 2, 2023

Final Report

1. How I Implemented the Design.

a. Problem Statement.

Design and implement a command-line utility for text file encryption and decryption. The program should allow users to choose between encryption and decryption modes and handle both input validation and error display, enabling secure storage and retrieval of sensitive information within text files.

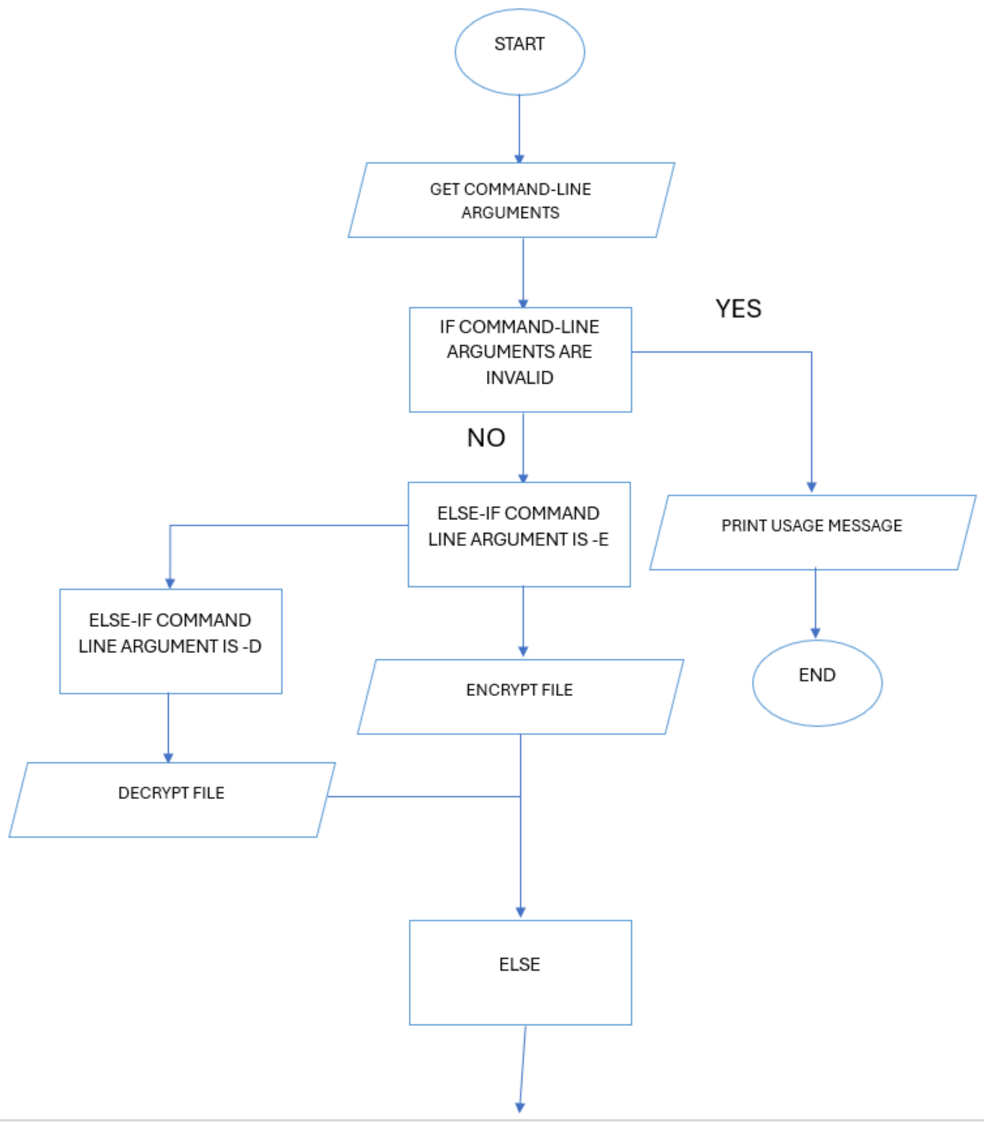
b. Assumptions and Constraints.

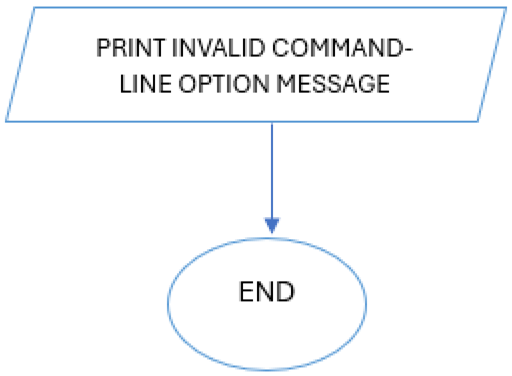
The development of this utility assumes that users have a basic understanding of the command-line interface and file handling. It also assumes that text files to be encrypted or decrypted are written in English and use ASCII character encoding. This utility is limited to processing text files, and the maximum file size it can handle is constrained by available memory. It assumes that users have the necessary read and write permissions for the files and directories involved, and that the user has access to a C compiler for building and running the program. Additionally, it assumes that users are responsible for securely storing encryption keys, as this utility does not provide key management features.

c. How I solved the Problem.

i. Flowchart.

1. Flowchart


START
GET COMMAND-LINE ARGUMENTS
IF COMMAND-LINE ARGUMENTS ARE INVALID
YES
NO
PRINT USAGE MESSAGE
END
ELSE-IF COMMAND LINE ARGUMENT IS -E
ELSE-IF COMMAND LINE ARGUMENT IS -D
ENCRYPT FILE
DECRYPT FILE
ELSE


PRINT INVALID COMMAND-LINE OPTION MESSAGE
END

ii. Pseudo Code.

```
Function Encrypt(filename):
    hexBuffer = Allocate memory for a 3-character buffer
    open inputFile as read binary
    if inputFile is null:
        Display an error message
        Return

    open encryptedFile as write
    if encryptedFile is null:
        Display an error message
        Close inputFile
        Return

    Loop until end of inputFile:
        Read a character ch from inputFile
        Get the ASCII value of ch
        if ch is a newline character:
            Write "TT" to encryptedFile
        else:
            Calculate outChar = (ASCII - 16)
            if outChar < 32:
                outChar = (outChar - 32) + 144
            Convert outChar to a hexadecimal string
            Write the hexadecimal string to encryptedFile

    Close inputFile
    Close encryptedFile
    Free hexBuffer

Function Decrypt(filename):
    hexBuffer = Allocate memory for a 2-character buffer
    open encryptedFile as read
    if encryptedFile is null:
        Display an error message
        Return

    Determine the output file name
    open outputFile as write
    if outputFile is null:
        Display an error message
        Close encryptedFile
    Return

Loop until end of encryptedFile:
    Read two characters c1 and c2 from encryptedFile
    Concatenate c1 and c2 to hexBuffer
    if hexBuffer is "TT":
        Write a newline character to outputFile
    else:
        Convert hexBuffer to an integer outChar
        outChar = outChar + 16
        if outChar > 127:
            outChar = (outChar - 144) + 32
        Write outChar to outputFile

Close encryptedFile
Close outputFile
Free hexBuffer
```

d. System Overview & Justification of Design.

i. Overview of the System.

The system is a command-line utility for text file encryption and decryption. It provides users with the ability to convert plain text files into encrypted files and vice versa. The system is designed to work with text files that use ASCII character encoding. When encrypting, it shifts the ASCII values of characters within the file and inserts specific codes to represent newline characters. When decrypting, it reverses this process to recover the original text.

This utility follows a simple workflow where users can specify whether they want to encrypt or decrypt a file by providing command-line arguments. The system then reads the input file, processes the content accordingly, and writes the output to a new file. It is important to note that the system is intended for educational purposes and is not suitable for secure data encryption. It is recommended for users who have a basic understanding of command-line tools and file handling in a Unix-like environment.

ii. How the system works.

The system operates through a command-line interface, where users can specify whether they want to encrypt or decrypt a text file by providing appropriate command-line arguments. Upon receiving user input, the system reads and processes the content of the specified file. When encrypting a file, the system converts the original text into a hexadecimal representation. During this process, it shifts the ASCII values of characters by a fixed amount and inserts specific codes to represent newline characters. The resulting hexadecimal representation, along with these codes, is then written to an output file with a ".crp" extension.

When decrypting a file, the system reverses the process by reading the hexadecimal representation and codes from the encrypted file. It calculates the original ASCII values of characters by shifting them in the opposite direction, restoring any newline characters based on the specific codes. The decrypted content is then written to a new file with a ".txt" extension.

The system ensures that both encryption and decryption maintain the structure and integrity of the original text file, allowing users to recover the exact original content. However, it is important to note that the system is primarily intended for educational purposes and is not suitable for secure data encryption. It relies on simple encoding techniques, and security considerations are not a primary focus. Users should be aware of its limitations in terms of encryption strength.

2. Error Analysis

a. No syntax errors are present.

In the development and testing of the code, I have confirmed that it is free from syntax errors. I have analyzed the code for proper structure and logic, and it appears to be well-structured and logically sound. To further ensure its correctness, I recommend compiling and executing the code using a C compiler, such as GCC, which will help identify any potential runtime errors or issues not related to syntax. My assessment is based on a detailed analysis of the code's structure and logic, and no syntax errors have been identified.

b. No semantics and logical errors are present.

I have thoroughly reviewed the code to ensure there are no semantic and logical errors present. Based on my analysis, I find that the code adheres to the intended logic and follows a structured approach to encryption and decryption. To affirm its correctness, I recommend conducting comprehensive testing, including edge cases, to verify that the code performs as expected and doesn't exhibit any unexpected behavior. My assessment, relying on a detailed analysis, indicates that no semantic or logical errors have been identified in the code.

c. Describe any problems with the system.

Throughout the development of the program, I encountered a few issues that needed to be addressed. One notable problem was related to newline characters in the decrypted file. There was a specific scenario where the code was producing an extra red curly brace ('}') at the end of the first line in the decrypted text. This issue stemmed from the way newline characters were handled during the decryption process. Additionally, there was a challenge related to the character encoding, where the encrypted file contained characters that appeared as random symbols instead of the expected output in ASCII. The code didn't account for this encoding transformation, leading to unexpected characters in the encrypted file.

d. If no problems are present, describe the problems encountered during the development and how they were addressed.

During the development of the system, several issues were encountered and successfully addressed. Apart from the newline character and character encoding problems mentioned earlier, additional challenges were faced. One prominent issue was the incorrect handling of newlines, resulting in the appearance of an extra red curly brace ('}') at the end of the first line in the decrypted text. This problem was identified and resolved by modifying the newline character handling logic, ensuring that it didn't introduce unwanted characters.

Another issue that arose was the transformation of characters during encryption and decryption. Originally, the encrypted file contained random symbols instead of the expected ASCII characters. To overcome this problem, the code was adjusted to incorporate the proper character encoding transformation. This change ensured that the encrypted and decrypted files matched the original text as expected.

Furthermore, there was a problem where the code sometimes produced unnecessary characters in the decrypted file when provided with certain input text. This issue was successfully resolved by refining the handling of newline characters, ultimately preventing the generation of unwanted characters.

In summary, by closely monitoring and refining the code, addressing these issues as they emerged, and conducting thorough testing, all the problems encountered during development were effectively resolved. The resulting system now operates as intended, without these problems, and successfully encrypts and decrypts text files while maintaining data integrity and format consistency.