CIS2201 Enigma machine simulator

Name: Caleb Bowden

Student ID: 2166829317

1. System design

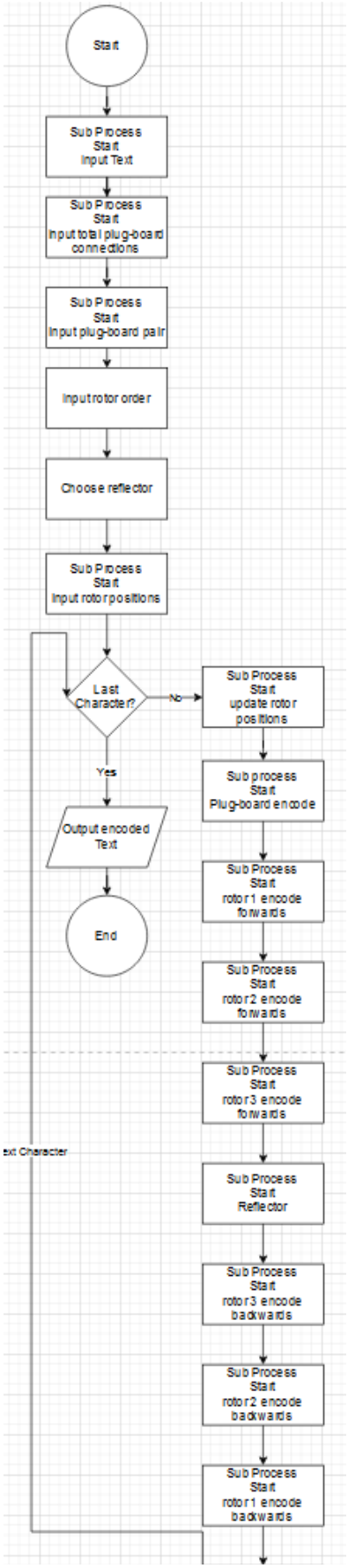


Figure 7 Main function the backbone of the code

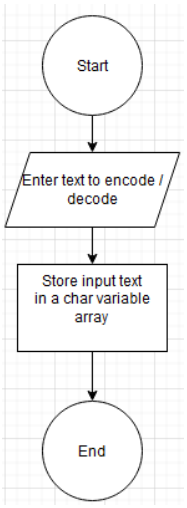


Figure 4 Input text to encode / decode function.

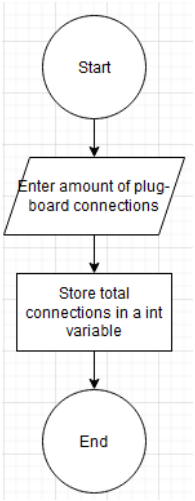


Figure 3 Input total plug-board connections function.

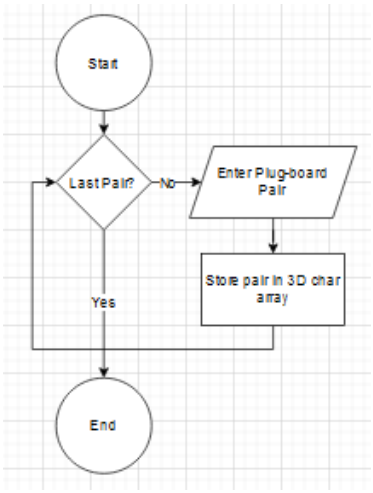


Figure 1 Input plug-board pair function.

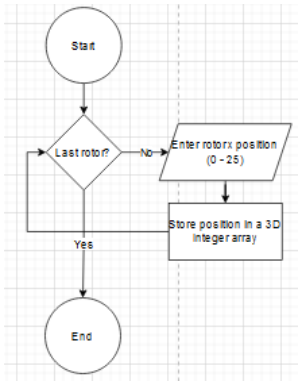


Figure 5 Input rotor start positions function.

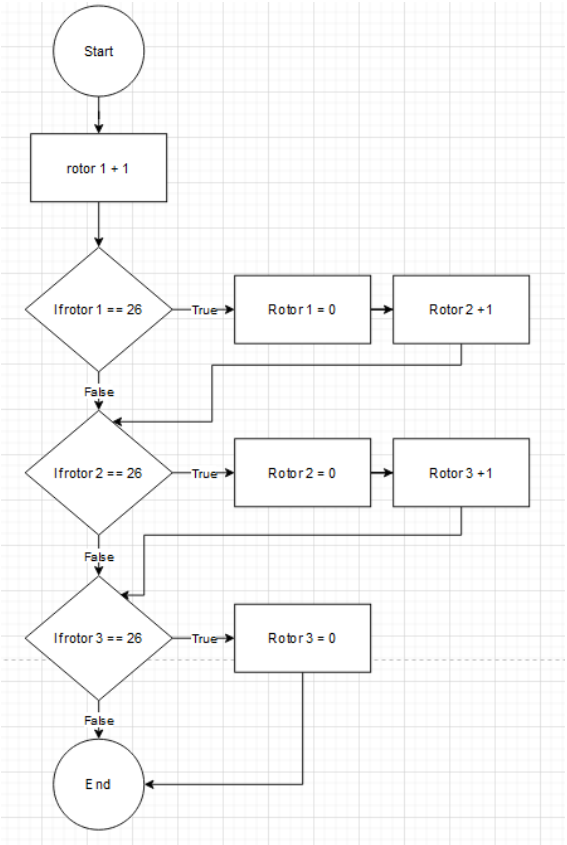


Figure 6 Update rotor positions function.

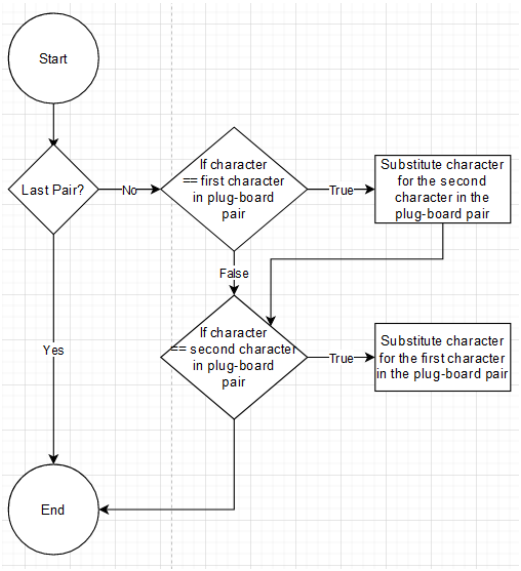


Figure 9 plug-board encode function.

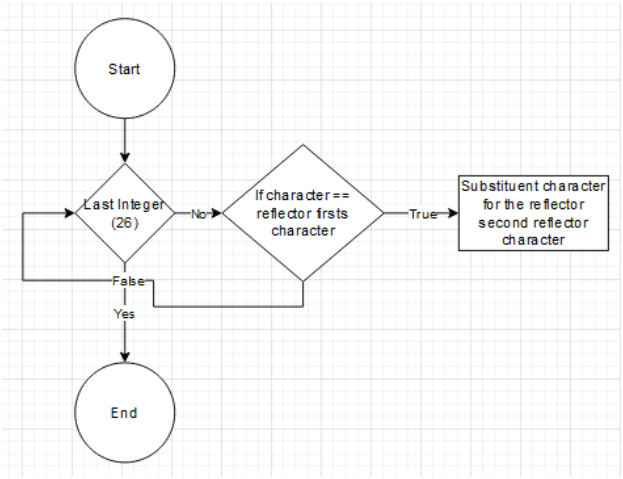


Figure 8 Reflector encode function.

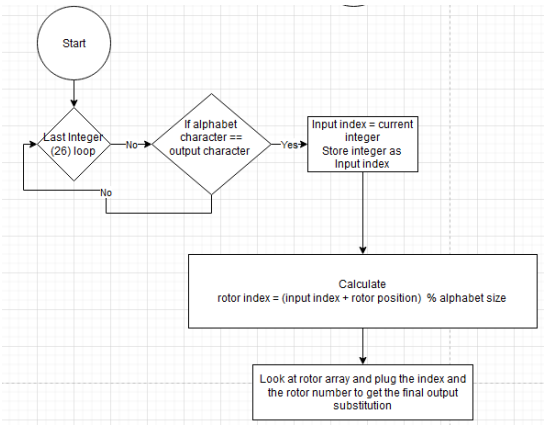


Figure 10 Encoding rotor forwards.

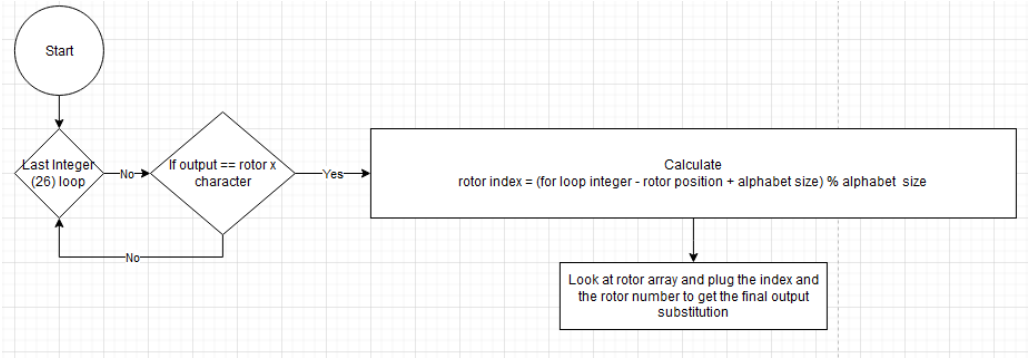


Figure 11 Encode character backwards.

For my system design, I opted to use a flowchart because of my familiarity with it and it ease in following while coding. In my flowchart, the main function serves as the backbone that connects all the sub-processes. This approach enhances readability and understanding of the system's flow. Each sub-process is compartmentalized to accomplish a specific task.

To begin, the input text processes collect the string entered by the user and store it in an array. The subsequent sub-process then prompts the user for the number of plug-board connections, which is stored in an integer variable. The next sub-process collects all the plug-board pairs, such as AB A <-> B bi-directional. Once all the user's information is obtained, the core functions of the enigma algorithm can commence.

In the main function, a loop iterates through each character of the input string, where the enigma algorithm executes. First, we call the rotor_position_update() function to update the rotor positions to ensure all rotors are correctly positioned. Next, the character is checked against the plug-board. If it is found in the plug-board, it is substituted for the corresponding character. The character then passes through three rotors, with each rotor encoding the character every time. The character then goes through the reflector, which is hard-mapped paired. If the output after the three rotors is A, the mapping for the reflector would output E. Once the character passes through the reflector, it goes back through the rotors in reverse order, from rotor 3 to rotor 1, encoding the character again to strengthen the encryption. Finally, the character goes back through the plugboard, resulting in a minimum of seven and a maximum of nine encryption times.

On reviewing my flowchart, I identified a few mistakes that could have been fixed. For instance, it would have been better to use the "for" loop with a "+1" to show the loop incrementing over time which would enhance the flowchart's readability.

I have changed a few things in the code that does not reflect in my design, an example of which is the reflector in the design it only shows 1 reflector however in my code you can change between reflector B & C. I also added a function where you can change the location of the rotors.

2. Implementation notes

a. Multiple words (scanf to fgets)

I encountered problems while using the scanf() function to store my input encode/decode string, specifically when attempting to store multiple words. I discovered that scanf() only stores the first word as it stops reading input when it encounters whitespaces. To resolve this, I researched alternative methods and found that fgets() reads the entire string, including whitespace characters. It requires three paraments to work. With fgets(), I was able to successfully store my entire string input.

b. Whitespace encryption

To encrypt each character of the input array, I created a for loop to iterate through the characters in the input array. Within the loop, I fed each character through the Enigma algorithm. However, I encountered an issue where whitespaces were being passed through the algorithm, resulting in random characters in the output. This created problems during the decryption process because the spaces were converted into letters, causing the rotors to move one extra place, and resulting in incorrect output. To fix this, I added an if-else statement within the for loop to check if the current character is a whitespace. If so, I skipped the Enigma algorithm for that character.

c. Rotor positions & movement

Initially, I believed that creating a rotor array and setting its initial position would allow the rotor to move forward by simply adding or subtracting steps. However, further research revealed that my understanding was flawed. Instead, to calculate the rotors position, you need to loop through the alphabet and +1 to the index until the input letter is reached. Adding the alphabet index to the rotor position and then using the mod operator to divide by 26 and getting the remainder. For moving backwards, the alphabet index is subtracted instead of adding & instead of looking at the alphabet array you look at the rotor array the reason we use the % sign is to get the remainder as this Is how you can update rotor positions and set them. This process determines the letter output of the rotor.

3. Testing

I have tested my developed code as follows:

Test 1: Here I am testing to see if my code can take in the string and encrypt it then decrypt the string to the original plain text.

```
"C:\Users\Admin\Desktop\Enigma Machine\Code\cmake-build-debug\EnigmaMachine.exe"
CAPITAL LETTERS ONLY
Enter text to encode / decode:HELLO

Enter number of plugboard pairs:1

Enter plugboard pairs (eg: AB A <-> B):
Pair 1:AB

Enter rotor order (0 = left, 1 = middle, 2 = right | rotors = 0 to 4):
Location 0 taken by rotor:4
 Location 1 taken by rotor:1
 Location 2 taken by rotor:3

Enter rotor start positions:
Rotor 4 (0-25):4
 Rotor 1 (0-25):2
 Rotor 3 (0-25):1

Choose reflector B or C (B = 0, C = 1):1

Output: AYMKC
Press any key to continue . . .
```

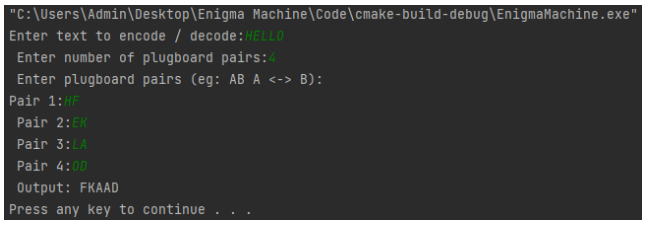
```
"C:\Users\Admin\Desktop\Enigma Machine\Code\cmake-build-debug\EnigmaMachine.exe"
CAPITAL LETTERS ONLY
Enter text to encode / decode:AYMKC

Enter number of plugboard pairs:1

Enter plugboard pairs (eg: AB A <-> B):
Pair 1:AB

Enter rotor order (0 = left, 1 = middle, 2 = right | rotors = 0 to 4):
Location 0 taken by rotor:4
 Location 1 taken by rotor:1
 Location 2 taken by rotor:3

Enter rotor start positions:
Rotor 4 (0-25):4
 Rotor 1 (0-25):2
 Rotor 3 (0-25):1

Choose reflector B or C (B = 0, C = 1):1

Output: HELLO
Press any key to continue . . .
```

Test 2: Here I am testing to see if my code performs correctly when it encounters no character i.e., a space.

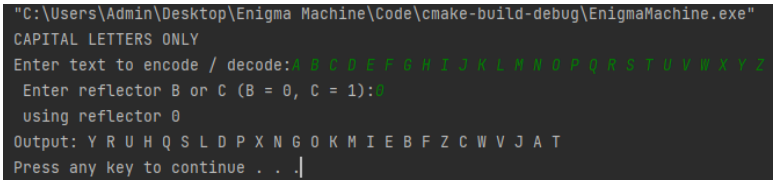
```
"C:\Users\Admin\Desktop\Enigma Machine\Code\cmake-build-debug\EnigmaMachine.exe"
CAPITAL LETTERS ONLY
Enter text to encode / decode:HELLO WORLD

Enter number of plugboard pairs:1

Enter plugboard pairs (eg: AB A <-> B):
Pair 1:AB

Enter rotor order (0 = left, 1 = middle, 2 = right | rotors = 0 to 4):
Location 0 taken by rotor:4
 Location 1 taken by rotor:0
 Location 2 taken by rotor:1

Enter rotor start positions:
Rotor 4 (0-25):4
 Rotor 0 (0-25):12
 Rotor 1 (0-25):19

Choose reflector B or C (B = 0, C = 1):1

Output: OKXEN KRMQZ
Press any key to continue . . .
```

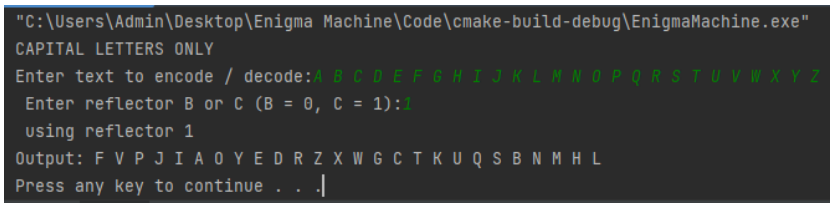
```
"C:\Users\Admin\Desktop\Enigma Machine\Code\cmake-build-debug\EnigmaMachine.exe"
CAPITAL LETTERS ONLY
Enter text to encode / decode:OKXEN KRMQZ

Enter number of plugboard pairs:1

Enter plugboard pairs (eg: AB A <-> B):
Pair 1:AB

Enter rotor order (0 = left, 1 = middle, 2 = right | rotors = 0 to 4):
Location 0 taken by rotor:4
 Location 1 taken by rotor:0
 Location 2 taken by rotor:1

Enter rotor start positions:
Rotor 4 (0-25):4
 Rotor 0 (0-25):12
 Rotor 1 (0-25):19

Choose reflector B or C (B = 0, C = 1):1

Output: HELLO WORLD
Press any key to continue . . .
```

Test 3: Here I am testing to see if my plugboard changes swaps letters correctly.

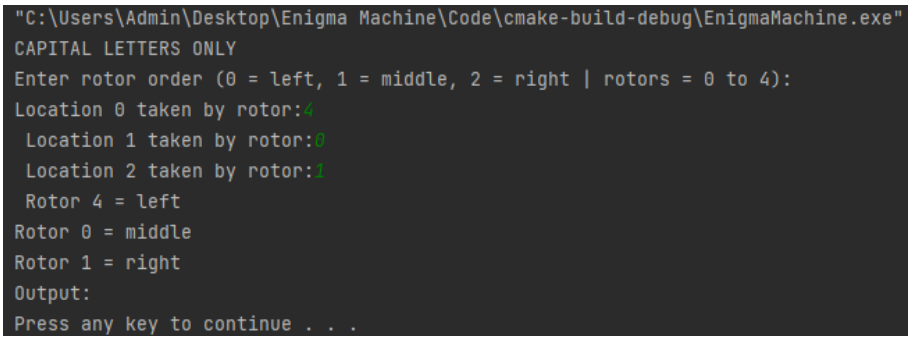


Test 4: Here I am testing to see if my reflector maps each character to another character.

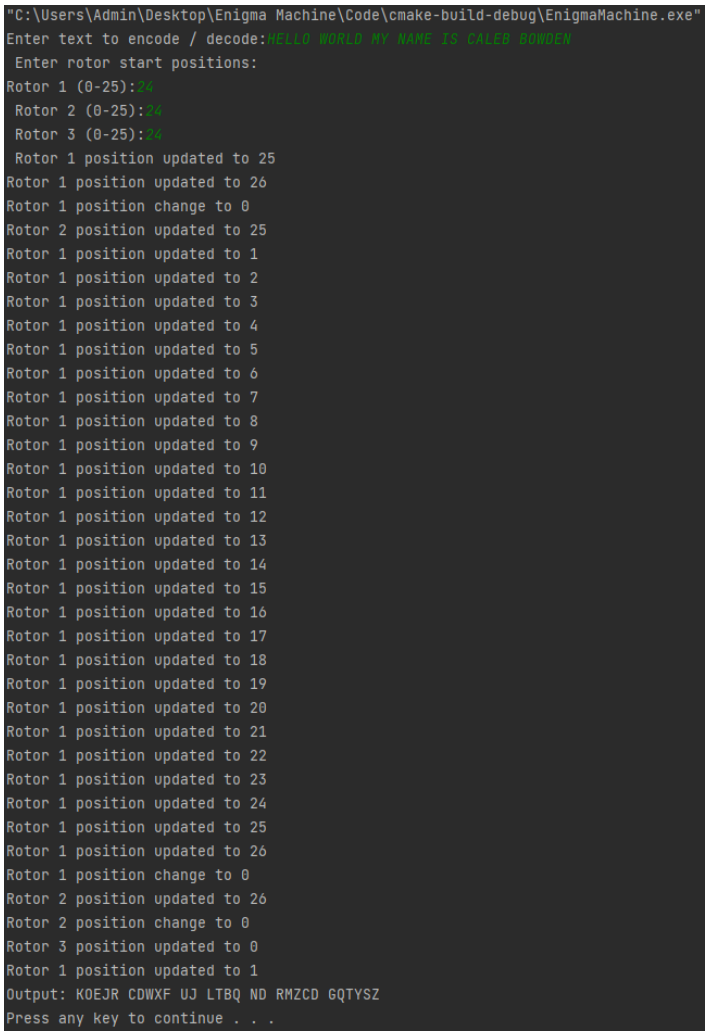




Test 5: Here I am checking to see if my rotor order code works correctly.



Test 6: Here I am testing to see if my rotor positions update correctly. And are set correctly.



Displaying 26 then updating to 0 because the code is counting from 1 not 0 when displaying numbers, this is not an error.

Test 7: Testing if my rotors substitute letters correctly when in the right position. (Forwards / & backwards)

```
"C:\Users\Admin\Desktop\Enigma Machine\Code\cmake-build-debug\EnigmaMachine.exe"
Enter text to encode / decode:HELLO
 Enter rotor start positions:
Rotor 1 (0-25):1
 Rotor 2 (0-25):1
 Rotor 3 (0-25):1
 rotor 1 (forwards) H -> V
rotor 2 (forwards) V -> F
rotor 3 (forwards) F -> C
rotor 3 (backwards) C -> F
rotor 2 (backwards) F -> V
rotor 1 (backwards) V -> H
rotor 1 (forwards) E -> D
rotor 2 (forwards) D -> S
rotor 3 (forwards) S -> A
rotor 3 (backwards) A -> S
rotor 2 (backwards) S -> D
rotor 1 (backwards) D -> E
rotor 1 (forwards) L -> Y
rotor 2 (forwards) Y -> E
rotor 3 (forwards) E -> L
rotor 3 (backwards) L -> E
rotor 2 (backwards) E -> Y
rotor 1 (backwards) Y -> L
rotor 1 (forwards) L -> H
rotor 2 (forwards) H -> X
rotor 3 (forwards) X -> Q
rotor 3 (backwards) Q -> X
rotor 2 (backwards) X -> H
rotor 1 (backwards) H -> L
rotor 1 (forwards) O -> P
rotor 2 (forwards) P -> Q
rotor 3 (forwards) Q -> W
rotor 3 (backwards) W -> Q
rotor 2 (backwards) Q -> P
rotor 1 (backwards) P -> O
Output: HELLO
Press any key to continue . . .
```

You can tell the rotors work correctly moving forwards and backwards because we get the same output as what we entered. when rotors go backwards it will follow the same path as it does not move position.

4. Reflective analysis

I believe that the enigma machine I created in C meets the requirements that were set. To ensure that my code is easily understandable, I utilized a modular approach by creating different functions for different aspects of the code. This not only enhances readability but also makes it easier to debug and modify the code in the future. Additionally, I took the time to comment my code thoroughly, explaining what each section does and how it fits into the larger program.

I am most proud of developing the rotor encode feature, both for its forward and backward functionalities. This task required a significant amount of time and effort, as I initially struggled to comprehend how to implement it. Through numerous iterations of the code, I eventually succeeded in creating a reliable encode/decode mechanism that met the project requirements.

There are a few things I would do differently. Firstly, I would have opted for a more authentic approach to encryption by asking the user to enter each letter individually & every 4 characters there is a break, as this would be closer to the original hardware enigma machine. Additionally, I would have improved my code by error checking user inputs to avoid any incorrect outputs or errors. Lastly, I would have explored multi-threading programming to speed up the encryption process. Overall, these changes would have made the project more accurate representation of the enigma machine.

Overall, I enjoy this project and it gave me a much better understanding of the C language.