Project 2: Enigma Machine

**Due** Mar 1 by 11:59pm | **Points** 50 | **Submitting** a file upload | **File Types** c | **Available** Feb 11 at 12am - Mar 1 at 11:59pm 19 days

**Corrections and Additions**

* Moved ROTOR\_CONSTANTS to enigma\_functions.c

**Files**

1. [enigma\_driver.c](https://canvas.wisc.edu/courses/280031/files/24618670?wrap=1)
2. [enigma\_functions.c](https://canvas.wisc.edu/courses/280031/files/24618672?wrap=1)
3. [enigma.h](https://canvas.wisc.edu/courses/280031/files/24618674?wrap=1)

**Learning Goals**

1. Learn how to write a C program that follows the given specifications.
2. Learn how functions improve code by making it modular and task-oriented.
3. Practice calling functions from different functions.
4. Practice passing arguments to functions.
5. Practice returning values from functions.
6. Develop code using one-dimensional and two-dimensional arrays.
7. Learn to write code that follows the commenting and the style guidelines.

**Encryption and Decryption Using the Enigma Machine**

Enigma machine uses rotors to encrypt messages. Inside the code for enigma\_driver.c there is an array of strings called ROTOR\_CONSTANTS that stores nine rotors. For our project, the rotor at position 0 is an identity rotor. It is a string with original English alphabets in the correct order.

Rotors at positions 1 - 8 of the array ROTORS are strings with the letters of the alphabet appearing in a shuffled order. For example, the rotors 0 to 3 are shown below.

Rotor 0 - ABCDEFGHIJKLMNOPQRSTUVWXYZ  
Rotor 1 - EKMFLGDQVZNTOWYHXUSPAIBRCJ  
Rotor 2 - AJDKSIRUXBLHWTMCQGZNPYFVOE  
Rotor 3 - BDFHJLCPRTXVZNYEIWGAKMUSQO

**How to use rotors to encrypt messages?**

Choose which rotors to use and in what order they will be applied. Rotate the letters in each rotor. For each letter of the message, find the position in the identity rotor and look up the letter at the same position in the encryption rotors. Then repeat with all of the other rotors.

**Encryption using two rotors without rotations**

Message: JAVA

Rotors to use: 31

Number of rotations: 0

Encrypted message:

Rotor 0 - ABCDEFGHI**J**KLMNOPQRSTUVWXYZ  
Rotor 3 - BDFHJLCPR**T**XVZNYEIWGAKMUSQO

The J is at position 9 in the identity rotor (rotor 0) so we find position 9 in rotor 3 and look up the letter T. Then we apply rotor 1.

Rotor 0 - ABCDEFGHIJKLMNOPQRS**T**UVWXYZ  
Rotor 1 - EKMFLGDQVZNTOWYHXUS**P**AIBRCJ

Look up the letter T in the identity rotor. It’s at position 19. Find position 19 in rotor 1 and and look up the letter. Repeat with the other letters.

Encrypted Message: PKOK

**Decryption**

To decrypt an encoded message we just apply the rotors in the reverse order.  Look up the letter in the encryption rotor, find its position then look up the letter at that position in the identity rotor.  Then repeat with the other rotors.  In the above example, we find the letter P in rotor 1 at position 19 then look up the letter at position 19 in the identity rotor, T.  Then apply rotor 3. Find the letter T and look up the letter at the same position in the identity

rotor.

**Rotating the rotors in Enigma**

Enigma machines also have the ability to change the letters in each encryption rotor by shifting all of the letters to the right. This is a cyclic shift.  For example, consider rotor 1 with 3 rotations applied to it.

Rotor 1 – EKMFLGDQVZNTOWYHXUSPAIBRCJ (before rotations)  
Rotor 1 – RCJEKMFLGDQVZNTOWYHXUSPAIB (after 3 rotations)

Notice the letter E begins in position 0 and after three rotations it is in position 3. Same with all of the other letters they are shifted down by 3 places.

After completing a rotation for the rotor configurations array, encryption and decryption procedures occur in the same way described before. When you want to encrypt a message with a particular number of rotations (for the rotors), then while decrypting that message back to its original message, you need to use the same value for the number of rotations (for the rotors) but the rotor indices should be used in the reverse order.

**Compiling and Running**

gcc -g enigma\_functions.c enigma\_driver.c -Wall  
./a.out

**Turn in**

Upload your enigma\_functions.c file to Canvas.

**Style**

Please follow the style guide linked on the Canvas Homepage.

**Logistics**

1. Do all of your work on the CSL instructional machines.
2. All assignments in this course are graded only on CS departmental machines running Linux Operating System. It is your responsibility to make sure that your code runs on these machines correctly.

**Debugging advice**

1. Write the comments first
2. Write your code in very small sections at a time and verify that it compiles before continuing.
3. Fix the compiler error messages from the top down. The first error may cause other compiler error messages to generate from the correct code later in your file.  So, scroll up and address the top error message first.
4. Add print statements
5. Use gdb to trace through your project
6. Ask questions about cryptic error messages. Post a screenshot of the error message on Piazza as a public post if you do not need to share any code. Post privately if you do need to share your code.