**Enigma Machine Project – Specification**

**Objectives**

1. Gain familiarity with basic OCaml language features, including built-in data types, lists, functions, and printing.

2. Practice writing programs in the functional style using immutable data.

3. Improve skills in computational thinking by learning about and modeling a mildly complex, real-world computational artifact.

**Requirements**

You are required to implement the following functions:

(\* [cipher refl rotors notches starts s] computes the Enigma cipher, where

- [refl] is the wiring of the reflector,

- [rotors] is a list of the rotors (which must contain at least one element), as they are installed from left to right on the spindle,

- [notches] is a list of where the notch is on each rotor, where the nth character of [notches] is the location of the notch on the nth rotor in [rotors],

- [starts] is a list of the starting character for each rotor as positioned on the spindle, where the nth character of [starts] is the starting character for the nth rotor in [rotors].

- [s] is the string to be ciphered. \*)

val cipher : string -> string list -> char list -> char list

-> string -> string

(\* [simulate] takes the same inputs as [cipher] but prints a simulation of the Enigma machine at each step of the computation. \*)

val simulate : string -> string list -> char list -> char list

-> string -> unit

**Wiring specifications:** The wiring of a rotor is specified as a 26-character string. The standard rotor I from the 1930 Enigma I, for example, would be "EKMFLGDQVZNTOWYHXUSPAIBRCJ". That means, in starting position 'A', when current is flowing right to left, the rotor would map 'A' to 'E', 'B' to 'K', 'C' to 'M', and so on. Of course, that map changes as the rotor steps. Two other standard rotors are rotor II, which is wired "AJDKSIRUXBLHWTMCQGZNPYFVOE", and rotor III, which is wired "BDFHJLCPRTXVZNYEIWGAKMUSQO".

Likewise, the wiring of a reflector is specified as a 26-character string. The standard reflector B would be "YRUHQSLDPXNGOKMIEBFZCWVJAT". That means that 'A' is mapped to 'Y', 'B' is mapped to 'R', 'C' is mapped to 'U', and so on. That map does not change.

**Stepping:** The intuition of stepping is that the rotors behave mostly like the numbers on an odometer: the rightmost completes a full revolution, and in so doing, causes the next rotor to its left to take a single step, and so forth. But the notch stepping system on the Enigma is a little more complicated:

*Rule 1:* The rightmost rotor always takes a single step just before enciphering each character.

*Rule 2:* Just before every character is enciphered, if any rotor *except the leftmost* is positioned at its notch, then that rotor and the rotor to its left steps.

*Rule 3:* No rotor steps more than once per character enciphered, even if the above rules could be construed as suggesting that a rotor would step twice.

Here is an example: Suppose the installed rotors are III-II-I, starting in position KDO. Then as each character is enciphered, they would step as follows: KDO, KDP, KDQ, KER, LFS, LFT, LFU, ...

Here is another example: Suppose the installed rotors are III-II-I, starting in position VDP. Then as each character is enciphered, they would step as follows: VDP, VDQ, VER, WFS, WFT, ...

**Testing**

Here are a few good test cases to get you started:

let id = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

let rotorI = "EKMFLGDQVZNTOWYHXUSPAIBRCJ"

let rotorII = "AJDKSIRUXBLHWTMCQGZNPYFVOE"

let rotorIII = "BDFHJLCPRTXVZNYEIWGAKMUSQO"

let reflB = "YRUHQSLDPXNGOKMIEBFZCWVJAT"

let () = assert ((cipher id [id] ['Q'] ['M'] "Q") = "Q")

let () = assert ((cipher reflB [rotorIII] ['Q'] ['Z'] "B") = "D")

let () = assert ((cipher id [id] ['Y'] ['Z'] "OCAML") = "OCAML")

let () = assert ((cipher reflB [rotorI; rotorII; rotorIII]

['Q'; 'E'; 'V'] ['F'; 'U'; 'N']

"OCAML") = "YNOXQ")

**Hints**

Think about implementing your solution in stages. Here is one possible progression:

* + First build something that works for messages that contain only a single character, with no stepping, with a single rotor that implements the wiring "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
  + Then improve your solution to something that works for other rotor wirings.
  + Then to multiple rotors.
  + Then to multiple characters. This is the hardest part, because it requires getting all the corner cases with stepping to work correctly. Remember, stepping always occurs just before a character is enciphered.