——advancingRotor——

These tests show how the rotor offsets get incremented in the correct order. The offsets correspond to [Left rotor, Middle rotor, Right rotor].

```
*Assignment2> :t advancingRotor
advancingRotor :: [Int] -> [Int]
*Assignment2> advancingRotor [0,0,0]
[0,0,1]
*Assignment2> advancingRotor [0,0,25]
[0,1,0]
*Assignment2> advancingRotor [0,25,25]
[1,0,0]
*Assignment2> advancingRotor [25,25,25]
[0,0,0]
*Assignment2> advancingRotor []
[]
```

——reflector——

These tests show that reflector takes in a character and returns the corresponding reflected character. It also shows that its reversible.

```
*Assignment2> :t reflector
reflector :: Char -> Char
*Assignment2> reflector 'D'
'H'
*Assignment2> reflector 'H'
'D'
*Assignment2> reflector 'A'
'Y'
*Assignment2> reflector 'Y'
'A'
```

——steckerboard——

Firstly, this test shows which letter corresponds to which. I them show how it is reversible, and if the letter isn't in the tuple list, or no list is supplied, the letter is returned.

```
*Assignment2> steckerTuple
[('A','B'),('C','D'),('E','F'),('G','H'),('I','J'),('K','L'),
('M','N'),('O','P'),('Q','R'),('S','T')]
*Assignment2> steckerboard 'A' steckerValues
'B'
*Assignment2> steckerboard 'B' steckerValues
'A'
*Assignment2> steckerboard 'K' steckerValues
'L'
*Assignment2> steckerboard 'L' steckerValues
'K'
*Assignment2> steckerboard 'L' []
```

```
'L'
*Assignment2> steckerboard 'U' steckerValues
'U'
```

——enigmaEncode——

Here I show enigma encode for the Simple Enigma, showing that its reversible and works with different offsets.

```
*Assignment2> enigmaEncode 'A' (SimpleEnigma [rotor1, rotor2,
rotor3] [1,2,3])
'W'
*Assignment2> enigmaEncode 'W' (SimpleEnigma [rotor1, rotor2,
rotor3] [1,2,3])
'A'
*Assignment2> enigmaEncode 'A' (SimpleEnigma [rotor1, rotor2,
rotor3] [25,25,25])
'U'
*Assignment2> enigmaEncode 'U' (SimpleEnigma [rotor1, rotor2,
rotor3] [25,25,25])
'A'
```

Here I show enigma encode for the Steckered Enigma, showing that its reversible and works with different offsets.

```
*Assignment2> enigmaEncode 'A' (SteckeredEnigma [rotor1, rotor2,
rotor3] [1,2,3] steckerValues)
'Q'
*Assignment2> enigmaEncode 'Q' (SteckeredEnigma [rotor1, rotor2,
rotor3] [1,2,3] steckerValues)
'A'
*Assignment2> enigmaEncode 'A' (SteckeredEnigma [rotor1, rotor2,
rotor3] [25,25,25] steckerValues)
‘F'
*Assignment2> enigmaEncode 'F' (SteckeredEnigma [rotor1, rotor2,
rotor3] [25,25,25] steckerValues)
'A'
```

——enigmaEncodeMessage——

Here I show enigma encode message for a simple enigma, showing that its reversible.

```
*Assignment2> enigmaEncodeMessage "HELLOWORLD" (SimpleEnigma
[rotor1, rotor2, rotor3] [5,6,25])
"VHGCGDFJOZ"
*Assignment2> enigmaEncodeMessage "VHGCGDFJOZ" (SimpleEnigma
[rotor1, rotor2, rotor3] [5,6,25])
"HELLOWORLD"
```

Here I show enigma encode message for a Steckered Enigma, showing that its reversible.

```
*Assignment2> enigmaEncodeMessage
"COMPUTERSCIENCE" (SteckeredEnigma [rotor4, rotor2, rotor1]
[5,25,0] steckerValues)
"BZVWCGWYAFBHONX"
*Assignment2> enigmaEncodeMessage
"BZVWCGWYAFBHONX" (SteckeredEnigma [rotor4, rotor2, rotor1]
[5,25,0] steckerValues)
"COMPUTERSCIENCE"
```

——crib——

My crib takes a message and a cipher and creates a triple with the letters at the same index

```
*Assignment2> testMessage
"WETTERVORHERSAGEBISKAYA"
*Assignment2> testCipher
"RWIVTYRESXBFOGKUHQBAISE"
*Assignment2> :t cribCreator
cribCreator :: Crib -> [(Int, String, String)]
*Assignment2> cribCreator testMessage testCipher
[(0,"W","R"),(1,"E","W"),(2,"T","I"),(3,"T","V"),(4,"E","T"),
(5,"R","Y"),(6,"V","R"),(7,"O","E"),(8,"R","S"),(9,"H","X"),
(10,"E","B"),(11,"R","F"),(12,"S","O"),(13,"A","G"),(14,"G","K"),
(15,"E","U"),(16,"B","H"),(17,"I","Q"),(18,"S","B"),(19,"K","A"),
(20,"A","I"),(21,"Y","S"),(22,"A","E")]
```

——menu & longest menu——

Currently my menu and longest menu don't work 100%, so far I can return the number of plain chars corresponding to the ciphered character at the position given. However this isn't recursive and I have to keep re-running the code with one of the outputted values, I feel I'm missing a very obvious recursion implementation, however no luck. Heres an example of what I mean:

This example is a very short menu, and I would like to print out [2,17].

```
*Assignment2> menu 2 testMessage testCipher
[17]
*Assignment2> menu 17 testMessage testCipher
[]
```

This example shows a longer menu, which I would have like to display as [1,2,5,21,12,7,4,2,17], as you can see, on the second run, 3 results were returned, and all three should be explored fully.

```
*Assignment2> menu 1 testMessage testCipher
[0]
*Assignment2> menu 0 testMessage testCipher
[5,8,11]
*Assignment2> menu 5 testMessage testCipher
[21]
*Assignment2> menu 21 testMessage testCipher
[12,18]
*Assignment2> menu 12 testMessage testCipher
[7]
*Assignment2> menu 7 testMessage testCipher
```

```
[1,4,10,15]
*Assignment2> menu 4 testMessage testCipher
[2,3]
*Assignment2> menu 2 testMessage testCipher
[17]
*Assignment2> menu 17 testMessage testCipher
[]
```

——general——

My simple enigma takes a list of strings (rotors) and a list of ints (offsets)
My steckered enigma takes a list of strings (rotors) and a list of ints (offsets) and a list of tuples with char, char (the steckered pattern).

```
*Assignment2> :t SimpleEnigma
SimpleEnigma :: [String] -> [Int] -> Enigma
*Assignment2> :t SteckeredEnigma
SteckeredEnigma :: [String] -> [Int] -> [(Char, Char)] -> Enigma
```

In the code, type Crib = (String, String) and type Menu = [Int]