**Bulbs**

**Pseudocode:**

-Prompt user to enter message.

-Turning message into decimal number.

- Covert decimal number into equivalent binary numbers (0,1).

\*one decimal 🡪 array size[8] = 0 and 1/ using loop< true = 1

<false = 0

\*loop: using %

-srart by right bit

-If numb. % 2 = 0 (no remaining) 🡪 turn off = 0

-If numb. % 2 != 0 (remaining) 🡪 turn on = 1

-Division is continued / 2 till reach final smal number

-Represent binary number by light bulb.

\*using print\_bulb turning 0, 1 🡪 on, of bulb

**Caser**

**Pseudocode:**

-Check program run with single command-line argument,

\*positive integer.

And if not, print a warning message (Usage: ./caesar key) and return 1.

\*if argc != 2 🡪 printf(“usage:./ceaser”) and return 1.

\*by for loop 🡪

if argv[1] is digit by func. 🡪 bool only\_digits(string s); TRUE

if argv[1] !digit 🡪 printf(“usage:./ceaser”) and return 1. FALSE

-Convert command-line argument from string to int.

\*by atoi func in stdlib.h (string to int). take key in int variable

-Prompt user for enter plaintext by get\_string.

-define array of characters which has size of strlen[plaintext]

-Iterate over each character of plain text.

\*by **for** loop:

-Check **if** letter or not by isalpha

-Check **if** islower 🡪encrypt as lower

-**else if** isupper 🡪encrypt as upper

**-else** not a character.

-Encipher alphabetic character.

\*convert ASCII into alphabetical index

(A = 65, a = 97// Z = 90, z = 122 ) 🡪 (A or a = 0 , Z = 25) HOW??

\*Shift alphabetical index using formula 🡪 Ci = (pi + k) % 26

-Index 🡪 (0🡪25)

\*Convert result back to ASCII

-Print Ciphertext

**Substitution**

**Pseudocode:**

- Get key by single command-line argument.

- Validate key

\*check key length 🡪 argc = 1 , character = 26

**else** error message “Usage: ./substitution key” and return 1.

\*check for non-alphabetic characters 🡪 only **isalpha** by for loop

**else** error message “The key must contain only letters.” and return 1.

\*check for repeated characters 🡪 no repeated characters **HOW??**

**Search:** create for loop and another one inside to check if is repeated **🡪**

error message “Key must not contain repeated character” and return 1.

\*if character != 26🡪 **else** error message“Key must contain 26 characters”

and return 1.

- Prompt user to enter plaintext by get\_string without a newline

\*only alphabetical are substituted 🡪

**- if** isalpha(plaintext) **else** ciphertext[ ] = plaintext[ ]

\*Lowercase letters remain lowercase, and uppercase letters remain uppercase. 🡪 ciphertext[ ] = isupper(Plaintext[ ])? ‘A’ : ‘a’ **??**

-Encipher

\*each character in the plaintext substituted for the corresponding character in the ciphertext; non-alphabetical characters should be outputted unchanged.

**HOW?**

**Thoughts:**

\*Create array for key [ 26 ] that user enter **&**  create array for plaintext[ ]

\*use index of key array as a pointer for plaintext character **HOW?**

\*Arranged letter of key = arranged letter of plaintext **HOW TO WRITE?**

\*plaintext[ ] 🡪 ciphertext **??**

\*preserve case 🡪 Lowercase letters remain lowercase, and uppercase letters remain uppercase.

\*leave non alphabetic characters as-is.

-Print chiphertext by printf and return 0.

\*use for loop to print chiphertext [i]

\*printf \n as newline

**Wordle**

**Pseudocode:**

-ensure the program accepts a single command-line argument.

\* if argc != 2 🡪 error message “Usage:./wordle wordsize” and return 1. **#TODO1**

\* provide a command-line argument, and make sure that **k** is one of the acceptable values (5, 6, 7, or 8), and store that value in wordsize but if it’s not in the correct rang 🡪

error message “wordsize must be either 5, 6, 7, or 8” and return 1. **#TODO2**

\* if the user provides a key of **n** 🡪 printf (“This is WORDLE50

You have 6 tries to guess the **n**-letter word I'm thinking of

Input a **n**-letter word: ”)

- making sure their guess is the correct length **#TODO3**

\*by function get\_guess

\*user should be prompted (as via get\_string) to type in a **k**-letter word,

**if** they supply a guess of the wrong length, they should be re-prompted

\*All guesses in this game should be in lowercase characters

- keep track of a user’s “score” in the game. **#TODO4**

\*In order to store those scores, we need an array, which we’ve calle status.