Assignment — ASCII Tools and Caesar Cipher (C Language)

In this lab, you will:

- Extend your previous **Caesar cipher** implementation.
- Add a new **Ascii structure** with function pointers simulating “methods” in C.
- Practice using **global variables**, **function pointers**, and **unit testing** with **Unity**.
- Write **unit tests** for both your cipher and your ASCII helper functions.

Step 1 — Add your previous Caesar cipher implementation

Copy your working Caesar cipher implementation from the previous assignment into caesar.c. Function prototype:

```
                int caesarCipher(char input_text[], int shift);
```

Step 2 — Extend your header file caesar.h

- Add the following structure definition **and global variables** to your header:

```
typedef struct
{
    char (*to_lower)(void);
    char (*to_upper)(void);
    int  (*is_digit)(void);
    int  (*is_letter)(void);
} Ascii;

extern char letter;
extern Ascii ascii;
```

- Then add prototypes for the initialization and ASCII utility functions:

```
•  void init_ascii(void);
•  void set_letter_ascii(char);
•  char to_lower_impl(void);
•  char to_upper_impl(void);
•  int  is_digit_impl(void);
•  int  is_letter_impl(void);
```

Step 3 — Implement the ASCII helper functions in caesar.c

All helper functions should operate on the global variable letter (declared in caesar.c).

```
char letter;
Ascii ascii;
```

Function descriptions:

- to_lower_impl(void)
  Converts letter to lowercase if it is an uppercase letter ('A'..'Z').
  Returns the lowercase letter, or **-1** if conversion is not possible.
- to_upper_impl(void)
  Converts letter to uppercase if it is a lowercase letter ('a'..'z').
  Returns the uppercase letter, or **-1** if conversion is not possible.
- is_digit_impl(void)
  Returns **1** if letter is a digit ('0'..'9'), otherwise **0**.
- is_letter_impl(void)
  Returns **1** if letter is a letter ('A'..'Z' or 'a'..'z'), otherwise **0**.

Step 4 — Initialize and use the structure

The following functions set up your global structure and letter variable:

```
void init_ascii(void)
{
    ascii.to_lower = to_lower_impl;
    ascii.to_upper = to_upper_impl;
    ascii.is_digit = is_digit_impl;
    ascii.is_letter = is_letter_impl;
}

void set_letter_ascii(char l)
{
    letter = l;
    return;
}
```

Example usage (for your main.c)

```
char tekst[] = "Hello world";
int result = caesarCipher(tekst, -50);
printf("%s\n", tekst);
init_ascii();
set_letter_ascii(tekst[0]);

printf("%d\n", ascii.is_digit());
printf("%d\n", ascii.is_letter());
printf("%c\n", ascii.to_lower());
printf("%c\n", ascii.to_upper());
```

This example illustrates:

- Linking the structure with implementation functions.

- Initializing the structure.
- Operating on the same letter variable using methods-like syntax:

```
•   ascii.is_letter();
•   ascii.to_upper();
```

Step 5 — Write Unit Tests (tests.c)

- Create **one test function per case**.
- Each function must be of type void (no parameters, no return value).
- Each test uses Unity macros (TEST_ASSERT_EQUAL, TEST_ASSERT_TRUE, etc.).

Tests for caesarCipher()

Test the following cases:

1. **Empty string**
   Input: ""
   Expected: no change, return 0.

2. **Null pointer**
   Input: NULL
   Expected: function returns -1.

3. **Shift key 2**
   Input: "abc" → Expected: "cde"

4. **Shift key -1**
   Input: "bcd" → Expected: "abc"

5. **Shift key 30 (larger than alphabet length)**
   Input: "abc" → Expected: "efg"

Tests for to_lower_impl() and to_upper_impl()

1. to_lower_impl() with 'A' → expected 'a'

2. to_upper_impl() with 'a' → expected 'A'

3. to_lower_impl() with '9' → expected -1

4. to_upper_impl() with '9' → expected -1

Tests for is_digit_impl() and is_letter_impl()

1. For '9' → is_digit() returns 1, is_letter() returns 0

2. For 'a' → is_digit() returns 0, is_letter() returns 1

Step 6 — Example test template

```
#include "unity.h"
#include "caesar.h"

void test_caesarCipher_empty_string(void)
{
    char text[] = "";
    int result = caesarCipher(text, 5);
    TEST_ASSERT_EQUAL(0, result);
    TEST_ASSERT_EQUAL_STRING("", text);
}

void test_to_lower_A(void)
{
    init_ascii();
    set_letter_ascii('A');
    TEST_ASSERT_EQUAL('a', ascii.to_lower());
}

void test_is_digit_9(void)
{
    init_ascii();
    set_letter_ascii('9');
    TEST_ASSERT_EQUAL(1, ascii.is_digit());
}
```

Step 7 — Summary

After completing this lab, you will understand:

- How to simulate “methods” in C using function pointers.
- How to share state between functions using global variables.
- How to test multiple independent functions using Unity.
- How to verify correctness of string transformations with unit tests.

Prototype (must match exactly):

int caesarCipher(char input_text[], int shift);

Behavior & requirements

1. The function **must:**
   - Check whether input_text is not NULL and not an empty string. If input is invalid, **return -1** (do **not** print any error message).
   - Convert any **uppercase letters** in the input to **lowercase** before shifting.

- Leave **non-alphabetic characters unchanged**.
- Apply the Caesar shift **in-place** to alphabetic characters only.
- Accept any integer shift (positive, negative, larger than 26). Normalise it so it wraps correctly around the 26-letter alphabet.
- Return **0** on success (i.e., when input_text was valid and processing completed), **-1** on error (invalid input_text).

2. **Important**: The function must **not** print anything (no printf for errors or results). Only the return code indicates success or error.