6.- `ft_strlen.-`

Function based on the definition given in the BSD man pages for "strlen".
The library associated is <string.h>.

Sinopsis: size_t strlen(const char *s);

Purpose: Calculates the length of a string.

Parameters:

- `s`: The string whose length is to be calculated.

Return value:

- The length of the string, which is the number of characters in the string, excluding the terminating null byte (`\0`).

Description:

The **`ft_strlen`** function calculates the length of the given string. It does this by iterating over the string character by character, counting each character until it reaches the terminating null byte (`\0`). The function returns the number of characters that werFunction based on the definition given in the BSD man pages for "strlen".

The library associated is <string.h>.

Sinopsis: size_t strlen(const char *s);e counted.

Code:

```
#include "libft.h"

size_t ft_strlen(const char *s)
{
    size_t i = 0;

    while (s[i] != '\0')
        i++;
    return (i);
}
/*
int main(void)
{
    char *s = "Hola Mundo";
    size_t i = ft_strlen(s);

    printf("Length of string \"%s\" is: %zu\n", s, i);
    return (0);
}
*/
```

Code explanation:

1. **Include header file:** The **`#include "libft.h"`** statement includes the header file **`libft.h`**, which defines the required libraries for our function.

2. **Define function:** The **`size_t ft_strlen(const char *s)`** statement defines the **`ft_strlen`** function. The function takes one argument, `s`, which is the pointer to the string whose length is to be calculated. The **`size_t`** data type is used to represent the size of the string, which is an unsigned integral type.
3. **Initialize counter:** The **`i = 0;`** statement initializes the counter variable `i` to 0. This variable will be used to keep track of the number of characters that have been counted.
4. **Iterate through string:** The **`while (s[i] != '\0')`** statement iterates over the string character by character. The **`s[i]`** expression evaluates to the current character in the string, and the `\0` expression evaluates to the null byte that terminates the string. The loop continues as long as the current character is not the null byte.
5. **Increment counter:** The **`i++;`** statement increments the counter variable `i` by 1. This is done after each character is counted, to keep track of the total number of characters that have been counted.
6. **Return length:** The **`return (i);`** statement returns the value of the counter variable `i`, which is the length of the string.
7. Under comments we develop a main function to show how it works:
   7.1. **Main function:** The **`int main(void)`** statement defines the main function, which is the entry point of the program.
   7.2. **char *s** : declares a pointer to a character.
   7.3. **size_t i**: declares a variable of type **`size_t`.**
   7.4 **Next steps:** we assign the string "Hola Mundo" to the pointer and call the **`ft_strlen`** function to get the length of the string.
   7.5 **Last step:** Finally, we print the length of the string and returns 0 to indicate successful completion of the program.

   7.5.1. Using %zu in printf: The **`printf("Length of string \"Hola Mundo\" is: %zu\n", i);`** statement prints a message indicating the length of the string `s`. The **`%zu`** format specifier is used to print the length of the string, which is a size_t data type. The **`%d`** format specifier would not be appropriate for this purpose, as it is intended for printing signed integer values.

   7.5.2. Reason for using %zu: The **`%zu`** format specifier is used because the **`size_t`** data type is unsigned, and the **`%d`** format specifier is intended for signed integer values. Using the **`%d`** format specifier on an unsigned integer value would result in undefined behavior.