7.- `ft_memset.-`

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

Sinopsis: void * memset(void *b, int c, size_t len);

Purpose: Fills a block of memory with a specific value.

Parameters:

- `b`: The memory area to be filled.
- `c`: The value to fill the memory with.
- `len`: The length of the memory area to be filled.

Return value:

- A pointer to the filled memory area.

Description:

The **`ft_memset`** function fills a block of memory with a specific value. It does this by iterating over the memory area character by character, assigning the specified value to each character. The function returns a pointer to the filled memory area.

The memset() function writes len bytes of value c (converted to an unsigned char) to the string b. The memset() function returns its first argument.

Code:

```
#include "libft.h"

void  *ft_memset(void *b, int c, size_t len)
{
      size_t      i;

      i = 0;
      while (i < len)
      {
            ((char *)b)[i] = (unsigned char)c;
            i++;
      }
      return (b);
}
/*
int   main(void)
{
      char  str[5];

      ft_memset(str, 'V', 4);
      str[4] = '\0';
      printf("%s\n", str);
      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 **`void *ft_memset(void *b, int c, size_t len)`** statement defines the **`ft_memset`** function. The function takes three arguments: **`b`**, **`c`**, and **`len`**. The **`b`** argument is a pointer to the memory area to be filled. The **`c`** argument is the value to fill the memory with. The **`len`** argument is the length of the memory area to be filled. The **`void *`** type is a generic pointer type that can point to any type of data.
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 filled.
4. **Fill memory with value:** The **`while (i < len)`** statement iterates over the memory area character by character. The **`((char *)b)[i]`** expression evaluates to the current character in the memory area, and the **`(unsigned char)c`** expression evaluates to the specified value to be filled. The loop continues as long as the current character is not the null byte (`\0`).
5. **Assign value:** The **`((char *)b)[i] = (unsigned char)c;`** statement assigns the specified value to the current character in the memory area. The **`(unsigned char)`** cast is necessary to ensure that the **`c`** argument is interpreted as an unsigned integer, as character values are stored in memory as unsigned integers.
6. **Increment counter:** The **`i++;`** statement increments the counter variable `i` by 1. This is done after each character is filled, to keep track of the total number of characters that have been filled.
7. **Return pointer:** The **`return (b);`** statement returns a pointer to the filled memory area. This pointer can be used to access the filled memory area later on.
8. Under comments we develop a main function to show how it works:

   8.1. Main function: The **`int main(void)`** statement defines the main function, which is the entry point of the program. The main function prompts the user to enter a string, reads the input string, and fills the string with the character 'V'. The **`printf("%s\n", str);`** statement prints the filled string.

   8.2. Input and fill string: The **`ft_memset(str, 'V', 4);`** statement fills the string **`str`** with the character 'V'. **The `str[4] = '\0';`** statement sets the last character of the string to null, indicating the end of the string.

   8.3. Return value: The **`return (0);`** statement exits the program with a status code of 0, indicating that the program executed successfully.