8.- `ft_bzero.-`

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

Sinopsis: void bzero(void *s, size_t n);

Purpose: Sets a block of memory to 0.

Parameters:

- `s`: The memory area to be set to 0.
- `n`: The length of the memory area to be set to 0.

Return value:

- None.

Description:

The **`ft_bzero`** function sets a block of memory to 0. It does this by iterating over the memory area byte by byte, assigning the value 0 to each byte. The function is a simple and efficient way to initialize a memory area to a known value.

The bzero() function writes n zeroed bytes to the string s. If n is zero, bzero() does nothing.

Code:

```
#include "libft.h"

void	ft_bzero(void *s, size_t n)
{
	unsigned char	*str;

	str = (unsigned char *)s;
	while (n-- > 0)
	{
		*str++ = 0;
	}
}
/*
int	main(void)
{
	char	str[11] = "Hello World";
	int		i = 0;

	ft_bzero(str, 3);
	while (i < 11)
	{
		printf("%c\n", str[i]);
		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 `void ft_bzero(void *s, size_t n)` statement defines the `ft_bzero` function. The function takes two arguments: `s` and `n`. The `s` argument is a pointer to the memory area to be set to 0. The `n` argument is the length of the memory area to be set to 0.
3. **Convert to unsigned char type:** The `str = (unsigned char *)s;` statement converts the pointer `s` to an unsigned char pointer. This is because the `ft_bzero` function operates on individual bytes of memory, and unsigned char is the data type used to represent a single byte.
4. **Iterate over memory area:** The `while (n-- > 0)` statement iterates over the memory area byte by byte. The `n--` statement decrements the `n` variable after each iteration, and the `> 0` condition ensures that the loop continues until `n` reaches 0.
5. **Assign value 0:** The `*str++ = 0;` statement assigns the value 0 to the current byte in the memory area. The `*str++` expression dereferences the `str` pointer and assigns the value 0 to the pointed-to byte. The `++str` statement increments the `str` pointer to point to the next byte in the memory area.
6. Under comments we develop a main function to show how it works:
   6.1. Main function: The `int main(void)` statement defines the main function, which is the entry point of the program. The main function creates a string `str` with the value "Hello World", sets the first 3 bytes of the string to 0, and prints the modified string.

   6.2. Fill string with 0: The `ft_bzero(str, 3);` statement sets the first 3 bytes of the string `str` to 0.

   6.3. Print modified string: The `while (i < 11)` statement iterates over the entire string `str`, printing each character.

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

New features:

1. **str = (unsigned char)s;:* This statement converts the pointer `s` to an unsigned char pointer. This is necessary because the `ft_bzero` function operates on individual bytes of memory, and unsigned char is the data type used to represent a single byte.
2. **while (n-- > 0):** This loop construct is a more concise way to write a loop that decrements a variable and continues as long as the variable is greater than 0. The `--` operator is used to

decrement the variable `n`, and the `>` operator is used to test the condition. This is equivalent to:

```
while (n--)
{
      if (n > 0)
      {
            *str++ = 0;
      }
}
```

3. The statement ***str++ = 0** is a combination of three separate operations: dereferencing, assigning, and incrementing. Let's examine each operation individually:

1. **Dereferencing:** The asterisk (*) symbol in front of `str` indicates that we are dereferencing the `str` pointer. Dereferencing a pointer means accessing the value that the pointer points to. In this case, `str` is pointing to a memory location, and dereferencing it gives us the actual value stored in that memory location.
2. **Assigning:** The equal sign (=) symbol indicates that we are assigning a value to the dereferenced pointer. In this case, we are assigning the value 0 to the memory location pointed to by `str`. This effectively sets the value of the byte to 0.
3. **Incrementing:** The double ampersand (++) symbol after `str` indicates that we are incrementing the `str` pointer. Incrementing a pointer means increasing the value stored in the pointer by 1. This effectively moves the pointer to the next byte in the memory area.

Putting it all together:

In summary, the statement **`*str++ = 0`** performs the following actions:

1. It retrieves the value stored at the memory location pointed to by `str`.
2. It assigns the value 0 to that location.
3. It increments the **`str`** pointer so that it points to the next byte in the memory area.

By combining these operations, we effectively set the value of each byte in the memory area to 0.