10.- ft_memmove.-

Function based on the definition given in the BSD man pages for “memmove(3)”.
The library associated is <string.h>.

Synopsis:

```
void *memmove(void *dst, const void *src, size_t len);
```

Purpose:

Copies a block of memory from one location to another, handling overlapping regions safely.

Parameters

- `dst`: Pointer to the destination memory block.
- `src`: Pointer to the source memory block.
- `len`: Number of bytes to copy.

Return Value

Returns a pointer to the destination memory block (`dst`).

8. Description

- Copies `len` bytes from `src` to `dst`.
- Safely handles overlapping regions by:
  - Copying from the beginning if `dst` is less than `src`.
  - Copying from the end if `dst` is greater than or equal to `src`.

The memmove() function copies len bytes from string src to string dst. The two strings may overlap; the copy is always done in a non-destructive manner.

The memmove() function returns the original value of dst.

9. Code

```
#include "libft.h"

void  *ft_memmove(void *dst, const void *src, size_t len)
{
	size_t	i;

	i = 0;
	if (len)
	{
		if (dst < src)
		{
			while (i < len)
			{
				((unsigned char *)dst)[i] = ((unsigned char *)src)[i];
				i++;
			}
		}
		else if (src < dst)
		{
			while (len --)
```

```
                        ((unsigned char *)dst)[len] = ((unsigned char *)src)
[len];
            }
      }
      return (dst);
}
```

Code Explanation

1. **Checks for overlapping regions:**
    - If `dst` is less than `src`, copies from the beginning (`i = 0`).
    - If `dst` is greater than or equal to `src`, copies from the end (`i = len - 1`).
2. **Copies bytes:**
    - Iterates through the required bytes (`len`).
    - Copies individual bytes from `src` to `dst` using casting and pointer arithmetic.
    - The code takes the addresses of `dst` and `src`, adds the offset `i` to them, and casts the resulting addresses to `unsigned char` pointers.
    - It then dereferences these pointers to access the individual bytes at those positions.
    - The assignment operator copies the byte from the source location (`src + i`) to the destination location (`dst + i`).
    - The parentheses ensure correct order of operations:
    - Casts are performed first, followed by pointer arithmetic, and finally, dereferencing.
    - This syntax enables precise byte-by-byte copying within the memory blocks, ensuring safe handling of overlapping regions.

Here it is how the parentheses works: `*(unsigned char *)(dst + i)`:

- `*`: Dereference operator, accessing the value stored at the memory address.
- `(unsigned char *)`: Casts the address (`dst + i`) to an `unsigned char` pointer.
- `(dst + i)`: Pointer arithmetic, adding `i` bytes to the starting address of `dst`.

3. **Returns `dst` pointer:**
    - Returns the pointer to the destination memory block.

Main Function (Optional)

```
int main(void)
{
    char src[18] = "memmove example";
    char dst[18];

    ft_memmove(dst, src, sizeof(src));
    printf("Copied string: %s\n", dst);
    return (0);
}
```

We use the above main function to show how memmove works. Printf shall give us the result of the string in dst; which has been copied byte by byte from src as defined by function.

ADDENDUM: (**few additional notes to enhance clarity)**

Key Points:

- **Memory Blocks:** Think of memory as a vast collection of tiny boxes, each holding a single byte of data.
- **Pointers:** They are like bookmarks, pointing to specific boxes in memory.
- **Overlapping Regions:** Imagine two blocks of memory partially covering each other, like overlapping puzzle pieces.

Casting:

- It's like translating a book from one language to another.
- Here, we're "translating" memory addresses to a common language of bytes (`unsigned char`) for consistent copying.

Pointer Arithmetic:

- It's like moving a bookmark forward or backward to access different boxes in memory.
- We use it to navigate through the bytes of the memory blocks.

Parentheses:

- They act like traffic signs, ensuring operations happen in the correct order.
- They enforce casting first, then pointer arithmetic, and finally dereferencing to access the bytes.

Analogy:

- Imagine copying a book word by word, being careful not to overwrite words you haven't copied yet.
- That's similar to how `ft_memmove` handles overlapping memory regions.

Remember:

- `ft_memmove` is designed for safe copying, even in tricky situations with overlapping memory.
- It's a valuable tool for working with memory in C programming.