28.- ft_itoa.-

While `ft_itoa` is a custom function not found in standard libraries, its functionality overlaps with various standard C functions. Here's a breakdown of relevant BSD man pages:

1. `strtol(3)`**:**

- Converts a string representation of a long integer to its numeric value.
- Similar to `ft_itoa` in converting a number to a string, but works in the opposite direction.
- Offers various base options (decimal, octal, hexadecimal) similar to `ft_itoa`'s potential use case.

2. `snprintf(3)`**:**

- Prints formatted output to a string buffer.
- Although not directly equivalent to `ft_itoa`, it can be used to format an integer into a string buffer, providing flexibility in formatting options.

3. `malloc(3)`**:**

- Allocates memory dynamically.
- Used by `ft_itoa` to allocate memory for the resulting string.
- Understanding `malloc` is crucial for memory management in dynamically allocated data like the converted string.

4. `free(3)`**:**

- Frees memory previously allocated with `malloc`.
- Essential for deallocating the memory used by the string created by `ft_itoa` to avoid memory leaks.

5. `strlen(3)`**:**

- Determines the length of a string.
- Although not directly used in `ft_itoa`, it might be helpful for understanding string manipulation and calculations related to string length.

Additional Notes:

- These man pages provide valuable information for understanding the underlying concepts and functions used in `ft_itoa`.
- While `ft_itoa` offers a custom implementation, these standard library functions serve as building blocks for working with strings and numbers in C.

Sinopsis: char *ft_itoa(int n)

Code:

```
#include "libft.h"

static int	ft_negative(int n)
{
	if (n < 0)
```

```
		return (-n);
	return (n);
}

static int	ft_nlen(int n)
{
	int	len;

	len = 0;
	if (n <= 0)
		len++;
	while (n != 0)
	{
		len++;
		n = n / 10;
	}
	return (len);
}

char	*ft_itoa(int n)
{
	char	*str;
	int		n_leng;

	n_leng = ft_nlen(n);
	str = malloc(sizeof(char) * n_leng + 1);
	if (!str)
		return (NULL);
	str[n_leng] = '\0';
	if (n < 0)
		str[0] = '-';
	if (n == 0)
		str[0] = '0';
	while (n != 0)
	{
		n_leng --;
		str[n_leng] = ft_negative(n % 10) + '0';
		n = n / 10;
	}
	return (str);
}
/*
int	main(void)
{
	int	s;

	s = 42;
	printf("Solution to evthg is: %s\n", ft_itoa (s));
}
*/
```

Explanation of the code:

1. **Include:** `#include "libft.h"`: This line includes the custom library "libft.h" which might contain definitions for used functions like `malloc`.
2. **Helper Functions:**
    - `ft_negative(int n):` This function takes an integer and returns its absolute value. It's used to handle negative numbers and convert them to positive for processing.
    - `ft_nlen(int n):` This function calculates the length of the string representation of the integer. It considers one extra character for the potential negative sign and counts digits till the number becomes zero.
3. **Main Function:**
    - `char *ft_itoa(int n):` This is the main function that takes an integer `n` as input and returns a null-terminated string representing it.
    - `n_leng = ft_nlen(n);:` This line calls the `ft_nlen` function to determine the length of the string needed.
    - `str = malloc(sizeof(char) * n_leng + 1);:` This line allocates memory for the string using `malloc`. It adds 1 to `n_leng` to accommodate the null terminator.
    - **Error Handling:** If memory allocation fails (`!str`), the function returns `NULL`.
    - **Null Terminator:** `str[n_leng] = '\0';` sets the last element of the array to the null terminator.
    - **Handling Negative Numbers:**
        - `if (n < 0)`: If the number is negative, the first character of the string is set to '-'.
    - **Handling Zero:**
        - `if (n == 0)`: If the number is zero, the first character of the string is set to '0'.
    - **Digit Conversion Loop:**
        - `while (n != 0)`: This loop iterates until the number becomes zero.
            - `n_leng --`: Decrement the string index for building the digits from the least significant to the most significant.
            - `str[n_leng] = ft_negative(n % 10) + '0';`: This line calculates the digit value:
                - `n % 10` gives the last digit of the number.
                - `ft_negative` handles negative numbers (converts to absolute value).
                - Adding '0' converts the numeric value to the corresponding ASCII character for the digit.
            - `n = n / 10;`: Divide the number by 10 to remove the processed digit for the next iteration.

    - **Return:** `return (str);` returns the pointer to the newly created string representing the integer.

Important Notes:

- This implementation assumes non-overflow conditions (the converted integer fits within the allocated memory).
- Remember to free the allocated memory after using the string using `free(str)`.

Example:

- In the `main` function, passing `42` to `ft_itoa` would result in a string "42" being returned.