29.- ft_strmapi.-

BSD Man Page References:

- `malloc(3)`: For memory allocation using `malloc`.
- `ft_strlen(3)` (custom, not standard): For calculating string length.

Synopsis:

```
char *ft_strmapi(char const *s, char (*f)(unsigned int, char));
```

Purpose:

Applies a user-defined function to each character of a string and returns a new string with the modified characters.

Description:

The `ft_strmapi` function takes two arguments:

- `s`: A pointer to the null-terminated string to be processed.
- `f`: A pointer to a function that takes two arguments:
  - `i`: An unsigned integer representing the index of the character in the string.
  - `c`: The character at the current index.
- The function `f` should return the modified character that will be placed in the new string.

The `ft_strmapi` function allocates memory for a new string with the same length as the original string plus one for the null terminator. It then iterates through each character of the original string, applies the function `f` to it, and stores the modified character in the new string. Finally, it adds the null terminator and returns the new string.

Code explanation:

```
char *ft_strmapi(char const *s, char (*f)(unsigned int, char))
{
    char *new_str;
    size_t i;

    // Check for null input
    if (!s || !f)
    {
        return (NULL);
    }

    // Allocate memory for the new string
    new_str = malloc(sizeof(char) * (ft_strlen(s) + 1));
    if (!new_str)
    {
        return (NULL);
    }

    // Apply the function to each character and build the new string
    i = 0;
    while (s[i])
    {
        new_str[i] = f(i, s[i]);
        i++;
    }

    // Add null terminator
    new_str[i] = '\0';

    // Return the new string
    return (new_str);
}
```

Example usage:

The provided `main` function demonstrates how to use `ft_strmapi` with a custom function called `my_function` that adds the index of each character to its ASCII value.

Additional notes:

- Remember to free the allocated memory for the new string using `free(new_str)` after you are finished using it.
- The function assumes that the user-defined function `f` does not modify the original string or allocate memory dynamically.

Main function added to explain this function explained:

```
char	my_function(unsigned int i, char c)
{
	return (c + (char)i);
}

int	main(void)
{
	char	*s = "Hello, world";
	char	*new_str;

	new_str = ft_strmapi(s, (char (*)(unsigned int, char)) & my_function);
	if (new_str == NULL)
	{
		printf("Error allocating memory\n");
		return (1);
	}
	printf("Original string: %s\n", s);
	printf("New string: %s\n", new_str);
	free(new_str);
	return (0);
}
```

Main Function Breakdown:

1. Purpose:

- Demonstrates how to use the `ft_strmapi` function with a custom function (`my_function`).
- Tests the functionality of `ft_strmapi` and showcases its output.

2. Step-by-Step Explanation:

- `char *s = "Hello, world";`: Declares a string pointer `s` and initializes it with the string "Hello, world".
- `char *new_str;`: Declares a character pointer `new_str` to store the modified string.
- `new_str = ft_strmapi(s, (char (*)(unsigned int, char)) & my_function);`:
  - Calls the `ft_strmapi` function with:
    - `s`: The original string.
    - `&my_function`: A pointer to the `my_function` function (casted to the correct type to match the `ft_strmapi` function signature).
  - The result is stored in `new_str`.
- `if (new_str == NULL)`: Checks if memory allocation failed. If so, prints an error message and returns 1.
- `printf("Original string: %s\n", s);`: Prints the original string.
- `printf("New string: %s\n", new_str);`: Prints the modified string returned by `ft_strmapi`.
- `free(new_str);`: Frees the memory allocated for the `new_str` to avoid memory leaks.
- `return (0);`: Indicates successful execution.

Key Points:

- The `main` function uses the `my_function` to add the index of each character to its ASCII value, resulting in a shifted string.
- It demonstrates how to pass a custom function pointer to `ft_strmapi`.
- It handles potential memory allocation errors and frees the allocated memory responsibly.