22.-`ft_calloc`.-

Not directly based on any BSD man page, but closely related to `calloc(3)`. Associated library is <stdlib.h>. (**ft_calloc: Dynamic Memory Allocation with Initialization)**

Synopsis: `void *calloc(size_t count, size_t size);`

Purpose:

- Allocates a contiguous block of memory and **initializes all bytes to zero**.
- Ideal for allocating arrays or structures where elements need to be explicitly set to 0.

Parameters:

- `count`: The number of elements to allocate in the block.
- `size`: The size (in bytes) of each element.

Return Value:

Returns a pointer to the allocated memory, or `NULL` if the allocation fails.

Description:

- Calls `malloc` to allocate the requested memory block.
- If allocation succeeds, uses `ft_bzero` to set all bytes in the block to zero (guaranteeing initial values are 0).
- Returns the pointer to the zero-initialized memory block.

Code:

```
#include "libft.h"

void *ft_calloc(size_t count, size_t size)
{
    void *rme;

    // Allocate memory using malloc
    rme = malloc(size * count);

    // Check if allocation was successful
    if (!rme) {
        return (NULL); // Return NULL if allocation fails
    } else {
        // Initialize all bytes to zero using ft_bzero
        ft_bzero(rme, size * count);
    }

    return (rme); // Return pointer to the allocated and initialized memory
}
```

Code Explanation:

1. **Memory Allocation:**
   - Calls `malloc(size * count)` to allocate memory for `count` elements of size `size`.
   - If `malloc` returns `NULL`, the allocation failed, so the function returns `NULL`.
2. **Initialization:**

        - If allocation succeeded, calls `ft_bzero(rme, size * count)` to set all bytes in the memory block pointed to by `rme` to zero.
        - This ensures that all elements, whether integer, floating-point, or other data types, start with a value of 0.
3. **Return Value:**
    - Returns the pointer `rme` to the allocated and initialized memory block.

Meaning and Usage of malloc:

`malloc` is a standard C library function that dynamically allocates memory on the heap. It takes the size of the memory block you want to allocate in bytes as its argument and returns a pointer to the beginning of the allocated block. If the allocation fails, it returns `NULL`.

However, using `malloc` alone doesn't initialize the memory to any specific value, leading to undefined behavior if you access uninitialized memory. This is where `ft_calloc` comes in, automatically initializing the allocated memory to zeros, making it safer and more convenient for specific use cases.

Key Points:

- `ft_calloc` uses `malloc` internally for memory allocation.
- It initializes all bytes of the allocated memory to zero using `ft_bzero`.
- Use `ft_calloc` when you need a block of memory where all elements must start at 0.
- Always remember to `free` the memory allocated by `ft_calloc` using `free` when you're done with it to avoid memory leaks.

Comments for the `main` Function:

```
int main(void)
{
    // Define the number of elements and element size
    size_t count = 5;
    size_t size = sizeof(int);

    // Allocate and initialize an array of 5 integers using ft_calloc
    int *arr = (int *)ft_calloc(count, size);

    // Check if allocation succeeded
    if (!arr) {
        printf("Error: Memory allocation failed.\n");
        return (1);
    }

    // Print the initial values (guaranteed to be 0)
    size_t i = 0;
    while (i < count) {
        printf("%d ", arr[i]);
        i++;
    }

    // Use the allocated array (e.g., assign new values)
    // ...
```