Dynamic Memory Allocation in C

Dynamic memory allocation refers to the process of allocating memory during the execution of a program (i.e., at runtime). It provides flexibility to manage memory as needed, unlike static memory allocation which reserves memory at compile time. In C programming, dynamic memory is allocated in the **heap** segment of memory.

To allocate and manage memory dynamically, C provides four standard library functions:

1. `malloc()` - Memory Allocation

- `malloc` stands for **memory allocation**.
- It is used to allocate a single block of uninitialized memory in the heap.
- It takes one argument: the size of memory to be allocated (in bytes).
- The memory allocated by `malloc` contains garbage values (whatever data was already at that memory location).

Syntax:

ptr = (data_type*) malloc(size);

2. `calloc()` - Contiguous Allocation

- `calloc` stands for **contiguous allocation**.
- It is used to allocate multiple blocks of memory and initialize all the bytes to zero.
- It takes two arguments: the number of elements and the size of each element.
- Since it initializes memory, `calloc` is generally slower than `malloc`.

Syntax:

ptr = (data_type*) calloc(number_of_elements, size_of_each_element);

3. `realloc()` - Reallocation

- `realloc` is used to resize the memory block that was previously allocated using `malloc` or `calloc`.
- It can increase or decrease the size of the allocated memory.
- It takes two arguments: a pointer to the previously allocated memory and the new size.

Syntax:

ptr = (data_type*) realloc(ptr, new_size);

4. `free()` - Deallocate Memory

- `free` is used to deallocate memory that was previously allocated using `malloc`, `calloc`, or `realloc`.
- It helps prevent **memory leaks**, which occur when allocated memory is not freed after use.

Syntax:

free(ptr);

Key Differences between `malloc()` and `calloc()`

Feature	malloc()	calloc()
Initialization	Uninitialized (garbage values)	Initialized to zero
Number of arguments	1	2
Speed	Faster	Slightly slower due to initialization

Additional Notes:

- Always check if the memory allocation was successful by checking if the returned pointer is `NULL`.

- It is good practice to free dynamically allocated memory once it's no longer needed.

Example:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *ptr;
    int n = 5;

    // Allocating memory using malloc
    ptr = (int*) malloc(n * sizeof(int));
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        return 1;
    }

    // Assigning values
    for (int i = 0; i < n; i++)
        ptr[i] = i + 1;

    // Displaying values
    for (int i = 0; i < n; i++)
        printf("%d ", ptr[i]);

    // Freeing memory
    free(ptr);

    return 0;
}
```

This concludes the basic explanation of dynamic memory allocation in C, with focus on `malloc`, `calloc`, `realloc`, and `free` functions.