# MEMORY MANAGEMENT IN C

***Puneet Saini, Ria Arora***
*Dronacharya College of Engineering*
*Haryana*

***Abstract :-*** **Abstract-Manual memory management for dynamic memory allocation in the C programming language is what is referred to as dynamic memory allocation.**
**There are two ways of allocating memory allocated in C, ie, by declaring the variables and by explicitly requesting space.**
**We have discussed variable declaration in other lectures, but here we will describe requesting dynamic memory allocation and memory management.**

**Space is allocated for a pointer by C whenever a pointer is declared.**
**For example: char *p;**
**Four consecutive bytes in memory are allocated which are associated with the variable p. Pointer to char is declared to be the type of pointer p. However, initialization of memory location does not happen, so it may contain garbage. Initialising the pointer at the time it is declared is usually a good idea. Its function is to reduce the chances of a random value in p to be used as a memory address. A pointer must be pointing to a valid area of memory before we can actually use it. The function malloc is used to request a pointer to a block of memory and calloc is used to request an array of zero-value initialized blocks.**

## I. INTRODUCTION

There are various functions in C for memory allocation and management, and these functions are found in the *stdlib* library.
malloc and calloc functions are used to reserve space in the memory. realloc is for moving a reserved block of memory to another allocation of different dimensions. free function is used for releasing space back to C. In clear words:

- To allocate space for an array in memory we use calloc()
- To allocate a memory block we use malloc()
- To reallocate a memory block with specific size realloc() is used
- To de-allocate previously allocated memory we use free()

## II. METHODOLOGY

A. malloc() Function:

During the execution of a program certain amount of memory is allocated to it with the help of malloc function. A block of memory from the heap is requested by the malloc function. The operating system will reserve the requested amount of memory if the request is granted. The user must return the memory to the operating system by calling the function free, if it is not required anymore.
An amount of memory with size of 32 bits or 4 bytes is asked for by the malloc statement. A NULL is returned if there is insufficient memory available. This is done by the malloc function. A block of memory is allocated, ie, it is reserved, if and only if the request is granted. The pointer variable places on itself the address of the reserved block.
The return value of NULL is checked for by the if statement. A message is printed and the program stops only if the return value equals NULL. A problem is interpreted if the return value of the program equals one. Another relief to the user is that he can make use of structures in a malloc statement. The structure may be used with or without a typedef statement.
SYNTAX:
ptr=(cast-type*)malloc(byte-size)
If we look at the dynamic memory functions, there are more functions of the stdlib.h library that the user can use to allocate dynamic memory. Four dynamic memory functions that can be found in the stdlib.h library are as follows:

B. calloc() Function:

The term calloc stands for "contiguous allocation". malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero. This is the only difference between calloc() and malloc() function.

Storage to variable is allocated using the calloc function while the program is running.This is done by writing calloc(num,size). Two arguments are

taken by this function which specify the number of elements to be reserved, and the size of each element in bytes and it allocates memory block equivalent to num * size . A pointer is returned by the function to the beginning of the allocated storage area in memory. The main difference between malloc and calloc function is that calloc initializes all bytes in the allocation block to zero and the allocated memory may/may not be contiguous.Space for dynamic arrays is reserved by calloc().

SYNTAX:
void * calloc (size_t n, size_t size);
The size in bytes of one element to the second argument is specified by the number of elements in the first argument. On successful partitioning, the address is returned, whereas NULL is returned on failure.
The parameters of the calloc() function are:

- **nitems**- It is the number of elements to be allocated.
- **size**- It is the size of elements.

C. relloc() Function:

Reallocation of a memory block with a specific new size is done by the realloc() function. The size of the memory block pointed to by the pointer is changed to the given size in bytes on calling the realloc() function. It is also responsible for the expansion and reduction of the amount of memory.
It is possible that the function moves the memory block to a new location. In this way the function will return this new location. The value of the new portion is indeterminate if the size of the requested block is larger than the previous block.
Even if the block is moved to a new location, the content of the memory block is preserved up to the lesser of the new and old sizes. The value of the newly allocated portion is indeterminate if the new size is larger.
The function will behave exactly like the function malloc() if the pointer is NULL. It will assign a new block of a size in bytes and will return a pointer to it.
The memory that was previously allocated is freed as if a call of the function free() was given if the size is 0. It will return a NULL pointer in such a case.
SYNTAX
void * realloc ( void * ptr, size_t size );
The realloc() function will return a pointer to the reallocated memory block. A NULL pointer is returned if the function fails.

D. free() Function:

The argument free() specifies the address of a dynamically allocated area. We can free the space using this function. When memory is allocated with either malloc() or calloc(), it is taken from the dynamic memory pool that is available to your program. This pool is finite and is sometimes called the heap. When a program finishes using a particular block of dynamically allocated memory, we should deallocate, or free, the memory to make it available for future use. We use free() function to free memory that was allocated dynamically.
SYNTAX:
void free(void *ptr);

This function releases the memory pointed to by ptr. The memory must have been allocated with malloc(), calloc(), or realloc(). No change happens if ptr is NULL.

**CONCLUSION**

We don't always know how much memory we will need to set aside at compile time. Imagine processing a data file (a series of calories burnt, say), where the number of records in the file isn't fixed. We could have as few as 10 records or as many as 100000. If we want to read all that data into memory to process it, we won't know how much memory to allocate until we read the file. And finally, dynamic memory allows us to build containers that can grow and shrink as we add or remove data, such as lists, trees, queues, etc. We could even build our own real "string" data type that can grow as we append characters to it (similar to the string type in C++).

## III. REFERENCES


- https://www.cs.cf.ac.uk/Dave/C/node11.html
- https://en.wikipedia.org/wiki/C_dynamic_memory_allocation
- https://en.wikipedia.org/wiki/C_(programming_language)
- https://en.wikipedia.org/wiki/Static_memory_allocation
- https://en.wikipedia.org/wiki/Memory_management#DYNAMIC
- https://en.wikibooks.org/wiki/C_Programming/Memory_management