Memory can be allocated for variables using different techniques

1. static allocation

decided by the compiler

allocation at load time [before the execution or run time]

2. automatic allocation

decided by the compiler

allocation at run time

allocation on entry to the block and deallocation on exit

3. dynamic allocation

code generated by the compiler

allocation and deallocation on call to memory allocation functions: malloc, calloc, realloc

Disadvantages of Static Memory Allocation

1. The memory is allocated during compilation time. Hence, the memory allocated is fixed and cannot be altered during run time.

2. This leads to under utilization of memory if more memory is allocated

3. This leads to over utilization of memory if less memory is allocated

4. Useful only when the data is fixed and known before processing

5. No deletion of the memory explicitly. Only overwriting effect can be seen.

6. Adding and deleting elements in between spends much time shifting.

Dynamic Memory Allocation

The process of allocating memory at runtime is known as dynamic memory allocation. Memory is allocated or deallocated during run-time (during the execution of the program) in the heap region. Library routines known as "memory management functions" are used for allocating and freeing/releasing memory during execution of a program. These functions are defined in stdlib.h

malloc(): Allocates requested size of bytes and returns a void pointer pointing to the first byte of the allocated space

calloc(): Allocates space for an array of elements, initialize them to zero and then return a void pointer to the memory

realloc(): Modifies the size of previously allocated space using above functions

free(): Releases the allocated memory

// to know in detail, look at programs

Advantages of Dynamic Memory Allocation

1. No wastage of memory when free is used on allocated memory

2. Allocated memory can be changed (expand or shrink) during run time of a program based on the dynamic requirement of the user/program.

Difference between SMA and DMA

SMA(Static Memory Allocation);

1. The memory is allocated before execution/run time

2. The size of the memory to be allocated is fixed during compile time and cannot be altered during run time.

3. Memory is allocated in stack and other segments

4. Used only when the data size is known in advance

DMA(Dynamic Memory Allocation):

1. The memory is allocated during run time

2. As and when memory is required, memory can be allocated. If memory is not required it can be deallocated.

3. Memory is allocated in heap area when functions are used.

4. Used for unpredictable memory requirement

**Functions in detail**:

malloc() :

Prototype: void \*malloc(size\_t size);

The malloc() function allocates size bytes and returns a pointer to the allocated memory. The memory is not initialized. If size is 0, then malloc() returns either NULL, or a unique pointer value. The malloc() function returns a void pointer to the allocated memory that is suitably aligned for any kind of variable. On error, these functions return NULL.

NULL may also be returned by a successful call to malloc() with a size of zero.

calloc()

prototype: void \*calloc(size\_t nmemb, size\_t size);

The calloc() function allocates memory for an array of nmemb elements of size

bytes each and returns a pointer to the allocated memory. The memory is set to zero. If nmemb or size is 0, then calloc() returns either NULL, or a unique pointer value. The calloc() function return a pointer to the allocated memory that is suitably aligned for any kind of variable. On error, this function return NULL.

NULL may also be returned by a successful call to calloc() with nmemb or size equal to zero.

realloc()

Prototype: void \*realloc(void \*ptr, size\_t size);

This function changes the size of the memory block pointed to by ptr to size bytes. The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes. If the new size is larger than the old size, the added memory will not be initialized. If ptr is NULL, then the call is equivalent to malloc(size), for all values of size; if size is equal to zero, and ptr is not NULL, then the call is equivalent to free(ptr). Unless ptr is NULL, it must have been returned by an earlier call to malloc(), calloc() or realloc(). If the area pointed to was moved, a free(ptr) is done.

The realloc() function returns a pointer to the newly allocated memory, which is suitably aligned for any kind of variable and may be different from ptr, or NULL if the request fails. If size was equal to 0, either NULL or a pointer suitable to be passed to free() is returned. If realloc() fails the original block is left untouched, and it is not freed or moved.

free()

void free(void \*ptr);

Description : The free() function frees the memory space pointed to by ptr, which must have been returned by a previous call to malloc(), calloc() or realloc(). Otherwise, or if free(ptr) has already been called before, undefined behavior occurs. If ptr is NULL, no operation is performed. The free() function returns no value.

Differences between malloc and calloc

malloc():

1. stands for memory allocation

2. This function takes single argument.

3. Syntax:

void \*malloc(size\_t size);

the required number of bytes to be allocated is specified as an argument.

4. Allocates a block of memory of size bytes

5. Allocated space will be initialized to undefined values

6. malloc is faster than calloc.

calloc():

1. Stands for contiguous allocation

2. This function takes two arguments

3. Syntax:

void \*calloc(size\_t n, size\_t size);

n is number of blocks to be allocated. size is number of bytes to be allocated for each block

4. Allocates multiple blocks of memory, each block with the same size

5. Each byte of allocated space is initialized to zero

6. calloc takes little longer than malloc because of the extra step of initializing the allocated memory by zero. However, in practice the difference in speed is very tiny and not recognizable.