Department of Computer Science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

MEMORY ALLOCATION

Abstract

Static and Dynamic Memory allocation

Vandana M Ladwani

vandanamd@pes.edu

Contents

Static and Dynamic Memory Allocation

Memory can be allocated for variables using two different techniques like static and dynamic memory allocation

Static Memory Allocation

Memory is assigned before the execution of the program begins i.e., during compilation time in the stack region. The compiler allocates the required memory space.

Advantages of Static Memory Allocation

1. Memory is allocated in a more structured area of memory, called the stack region.
2. No need of pointers to access the data
3. Faster execution than Dynamic memory allocation

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. Memory cannot be deleted explicitly only overwriting takes place
6. If elements are to be added to or deleted from intermediate positions then shifting of elements is required to be done

Dynamic Memory Allocation

Consider a particular application for which we need to change the size of array at run time, if array is allocated statically, extending or shrinking of the memory during run time is not possible. Thus by using static memory allocation, we cannot utilize the memory efficiently instead we can use dynamic memory allocation in this scenario

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 defined in **stdlib.h** known as "memory management functions" are used for allocating and releasing also termed as freeing memory during execution of a program. These functions are as follows

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

Advantages of Dynamic Memory Allocation

1. Memory is allocated only when the program unit is active.
2. Memory is utilized efficiently.
3. Allocated memory can be resized during run time based on the dynamic requirement of the user/program.

Disadvantages of Dynamic Memory Allocation

1. Memory is allocated in the heap region which is less structured.
2. Execution is comparatively slower.
3. Pointers are required to work with dynamically allocated memory region.

Difference between static and Dynamic Memory Allocation

Static Memory Allocation)	Dynamic Memory Allocation
The memory is allocated during compile time.	The memory is allocated during run time
The size of the memory to be allocated is fixed during compile time and cannot be altered during run time.	As and when memory is required, memory can be allocated. If memory is not required it can be deallocated.
Memory is allocated in stack area	Memory is allocated in heap area
Used only when the data size is fixed and known in advance before processing	Used for unpredictable memory requirement
Execution is faster, since memory is already allocated and data manipulation is done on these allocated memory locations	Execution is slower since memory has to be allocated during run time. Data manipulation is done only after allocating the memory
Example: Arrays	Example: Linked List

Dynamic Memory Management Functions

malloc()

Prototype: void *malloc(size_t size);
Description: 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. On error, function return NULL.

calloc()

Prototype: void *calloc(size_t num_blocks, size_t size);
Description: The calloc() function allocates memory for an array of num_blocks elements of size bytes each and returns a pointer to the allocated memory. The memory is set to zero. If num_blocks or size is 0, then calloc() returns either NULL, or a unique pointer value. The calloc() function return a void pointer to the allocated memory.

realloc()

Prototype: void *realloc(void *ptr, size_t size);
Description: 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 size>0; if size is equal to zero, and ptr is not NULL, then the call is equivalent to free(ptr). If the area pointed by ptr gets 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 realloc() fails the original block is left untouched, and it is not freed or moved.

free()

Prototype: 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 behaviour occurs. If ptr is NULL, no operation is performed. The free() function returns no value.

Differences between malloc and calloc

malloc()	calloc()
stands for memory allocation	Stands for contiguous allocation
This function takes single argument. Syntax: void *malloc(size_t size); size: total number of bytes to be allocated	This function takes two arguments Syntax: void *calloc(size_t n, size_t size); n :number of blocks to be allocated. Size: number of bytes to be allocated for each block
Allocates a block of memory of size bytes	Allocates multiple blocks of memory, each block with the same size
Allocated space will be initialized to junk values	Each byte of allocated space is initialized to zero
malloc is faster than calloc.	calloc takes little longer(not recognizable in practical scenario) than malloc because of the extra step of initializing the allocated memory.