Memory Allocation in C

1. Static Memory Allocation
2. Dynamic Memory Allocation

Static Memory Allocation: When the allocation of memory performs at the compile time, then it is known as static memory. In this, the memory is allocated for variables by the compiler.

Dynamic Memory Allocation

When the memory allocation is done at the execution or run time, then it is called dynamic memory allocation.

Difference between Static and Dynamic Memory Allocation

Static Memory Allocation	**Dynamic Memory Allocation**
When the allocation of memory performs at the compile time, then it is known as static memory.	When the memory allocation is done at the execution or run time, then it is called dynamic memory allocation.
The memory is allocated at the compile time.	The memory is allocated at the runtime.
In static memory allocation, while executing a program, the memory cannot be changed.	In dynamic memory allocation, while executing a program, the memory can be changed.
Static memory allocation is preferred in an array.	**Dynamic memory allocation is preferred in the linked list.**
It saves running time as it is fast.	It is slower than static memory allocation.
Static memory allocation allots memory from the stack.	Dynamic memory allocation allots memory from the heap.
Once the memory is allotted, it will remain from the beginning to end of the program.	Here, the memory can be alloted at any time in the program.
This memory allocation is simple.	This memory allocation is complicated.
Static memory allocation is less efficient as compared to Dynamic memory allocation.	Dynamic memory allocation is more efficient as compared to the Static memory allocation.

Dynamic Memory Allocation in C

Need for Dynamic Memory allocation in C:

An array is collection of items stored at continuous memory locations. int a[10];

length of an array: 10

But what if there is a requirement to change this length (size). For Example,

If there is a situation where only 5 elements are needed to be entered in this array. In this case, the remaining 5 indices are just wasting memory in this array.

- So, there is a requirement to lessen the length (size) of the array from 10 to 5.

Take another situation. In this, there is an array of 10 elements with all 0 to 9 indices filled. But there is a need to enter 3 more elements in this array. In this case 3 indices more are required. So the length (size) of the array needs to be changed from 10 to 13.

This can be done with the help of Dynamic Memory Allocation in C. Therefore, C Dynamic Memory Allocation can be defined as a procedure in **which the size of a data structure (like Array) is changed during the runtime**. C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under **<stdlib.h>** header file to facilitate **dynamic memory allocation in C programming. They are:**

1. malloc()
2. calloc()
3. free()
4. realloc()

malloc()

“malloc” or “memory allocation” method in C is used to dynamically allocate a single large block of memory with the specified size. **It returns a pointer of type void which can be cast into a pointer of any form.**

Syntax:
ptr = (cast-type*) malloc(byte-size)
ptr = (cast-type*) malloc(no. of elements * size of data type)

For Example:
ptr = (int*) malloc(100 * sizeof(int));

Since the size of int is 2 bytes, this statement will allocate 200 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

// program using malloc ()

```
#include  <stdio.h>
#include <stdlib.h>
void main()
{
// This pointer will hold the  base address of the block created
int*    ptr;
int n, I ;
printf("enter      no      of      elements");
scanf("%d",&n);
// Dynamically allocate memory using malloc()

ptr = (int*)malloc(n * sizeof(int));

// Check if the memory has been successfully  allocated by malloc or not
if (ptr == NULL)
{
        printf("Memory not allocated.\n");
        exit(0);
}
else
{
// Get the elements of the array
for (i = 0; i < n; ++i)
          {
            scanf("%d",ptr+i);
          }
// Print the elements of the array
printf("The elements of the array are: "); for
(i = 0; i < n; ++i)
{
printf("%d, ", ptr[i]);
}
}
free(ptr);
}
```

calloc()

“calloc” or “contiguous allocation” method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value ‘0’.

Syntax:
ptr = (cast-type*)calloc(n, element-size);
For Example:
ptr = (float*) calloc(25, sizeof(float));

This statement allocates contiguous space in memory for 25 elements each with the size of the float.

```
// calculate the sum of n numbers using calloc()
#include  <stdio.h>
#include <stdlib.h>
void main()
{
// This pointer will hold the  base address of the block created
int* ptr;
int n, i, sum = 0;
printf("enter no of elements");
scanf("%d",&n);
// Dynamically allocate memory using calloc()

ptr = (int*) calloc(n, sizeof(int));

// Check if the memory has been successfully  allocated by calloc or not
if (ptr == NULL)
{
printf("Memory  not  allocated.\n");
exit(0);
}
else
{
// Get the elements of the array
for (i = 0; i < n; ++i)
{
                scanf("%d",ptr+i);
                sum=sum+ptr[i];
}
printf ("\nsum of elements are \t%d",sum);
}
free(ptr);
}
```

free()

"free" method in C is used to dynamically de-allocate the memory. The memory allocated using functions malloc() and calloc() is not de-allocated on their own. Hence the free() method is used, whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

Syntax:
free(ptr);

realloc()

"realloc" or "re-allocation" method in C is used to dynamically change the memory allocation of a previously allocated memory. In other words, if the memory previously allocated with the help of malloc or calloc is insufficient, realloc can be used to dynamically re-allocate memory.
Syntax:
ptr = realloc(ptr, newSize);
where ptr is reallocated with new size 'newSize'.

```
//program on realloc() and free()
#include  <stdio.h>
#include <stdlib.h>
int main()
{
int* ptr;
int n, n1,i, sum = 0;
printf("enter    the    number    of    elements");
scanf("%d",&n);
// Dynamically allocate memory using calloc()

ptr = (int*)calloc(n, sizeof(int));

if (ptr == NULL)
{
printf("Memory not allocated.\n");
exit(0);
}
else
{

for (i = 0; i < n; ++i)
 {
scanf("%d",ptr+i);
}
printf("enter the new size of an array");
scanf("%d",&n1);

ptr = realloc(ptr, n1* sizeof(int));

for (i =n ; i < n1; ++i)
{
scanf("%d",ptr+i);
}
// Print the elements of the array
printf("The elements of the array are: "); for
(i = 0; i < n1; ++i) {
printf("%d, ", ptr[i]);
}
free(ptr);
}
return 0;
}
```

2017-18 (RCS-101)

1. Define the concept of pointer? Also define the dynamic memory allocation and various functions for dynamic memory allocation, with suitable examples. 7

2018-19(KCS-101)

1. What is dynamic memory allocation? Explain the calloc(), malloc(), realloc() and free() functions in details. What is lifetime of a variable, which is created dynamically. 10

2018-19(KCS-201)

1. Differentiate static and dynamic memory allocation. 2

2019-20(KCS-101)

1. State the features of a pointer. Explain dynamic memory allocation with the help of an example. 10

2020-21(KCS-101T)

1. Explain dynamic memory allocation concept with proper example. 10

2021-22(KCS-101T)

1. Define dynamic memory allocation. Differentiate between malloc () and calloc () with proper example. 10

2021-22(KCS-201T)

1. What is the requirement of FREE() function in Dynamic memory allocation. 2
2. Write a program to allocate the memory with dynamic memory allocation concept. Take the input from the user and find the sum of all elements. 10

2022-23(BCS-101)

1. Discuss dynamic memory allocation. Explain calloc(), malloc(), realloc() and free() functions with suitable example. 7

2022-23(BCS-201)

1 Explain static memory allocation and dynamic memory allocation with suitable examples.

2023-24(BCS-101)

1 Define calloc function. Write the program to print the sum of elements initialized at the dynamic memory allocated by the user. 7