Dynamic Memory Allocation in C

Before learning Dynamic Memory allocation, let's understand first: How the memory management is done C?

When we declare a variable using a basic data type, the C compiler automatically allocates memory space for the variable in a pool of memory called the **stack**. For example, a float variable takes typically 4 bytes when it is declared. Also, an array with a specified size is allocated in contiguous blocks of memory, where each block has the size for one element. This can be verified using the **sizeof** operator.

As learned so far, when declaring a basic data type or an array, the memory is automatically managed. However, there is a process for allocating memory which will permit you to implement a program in which the array size is undecided until you run your program (runtime). This process is called "**Dynamic memory allocation**."

Dynamic Memory Allocation

Dynamic Memory Allocation is manual allocation and freeing of memory according to your programming needs. Dynamic memory is managed and served with pointers that point to the newly allocated memory space in an area which we call the heap.

Now we can create and destroy an array of elements dynamically at runtime without any problems. To sum up, the automatic memory management uses the stack, and the dynamic memory allocation uses the heap.

The C programming language provides several functions for memory allocation and management. These functions can be found in the **<stdlib.h>** header file. Dynamic memory allocation in C language is possible by 4 functions of stdlib.h header file.

Function	Purpose
malloc	Allocates the memory of requested size and returns the pointer to the first byte of allocated space.
calloc	Allocates the space for elements of an array. Initializes the elements to zero and returns a pointer to the memory.
realloc	It is used to modify the size of previously allocated memory space.
Free	Frees or empties the previously allocated memory space.

malloc() function in C

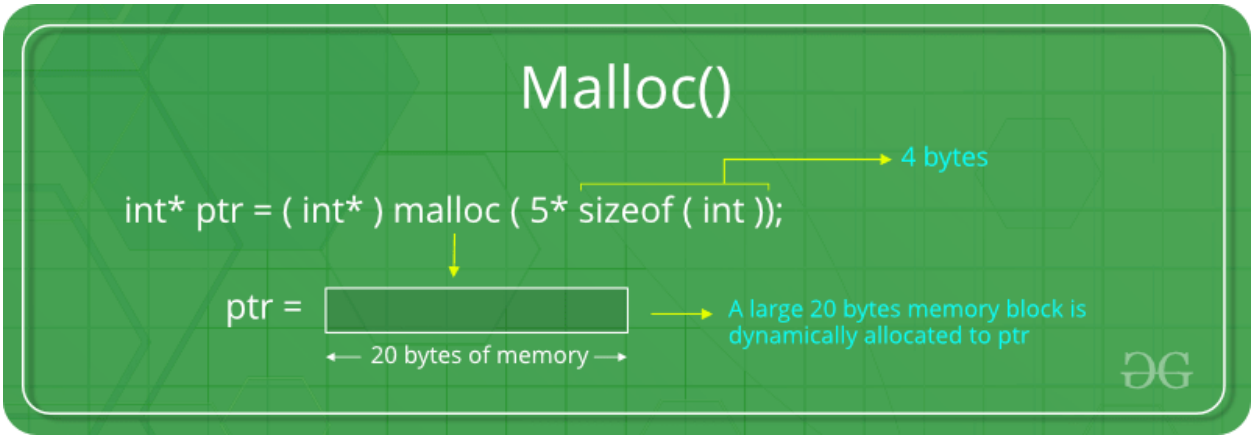
"malloc" or **"memory allocation"** method in C is used to dynamically allocate a single large block of memory with the specified size. It doesn't initialize memory at execution time, so it has garbage value initially. It returns NULL if memory is not sufficient. The syntax of malloc() function is given below:

ptr=(cast-type*)malloc(byte-size);

For Example:

ptr = (int*) malloc(100 * sizeof(int));

Since the size of int is 4 bytes, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.



Example

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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", 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 {
```

```
        // Memory has been successfully allocated
        printf("Memory successfully allocated using malloc.\n");

        // Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }

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

    return 0;
}
```

OUTPUT

```
Enter number of elements: 5
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5,
```

calloc() function in C

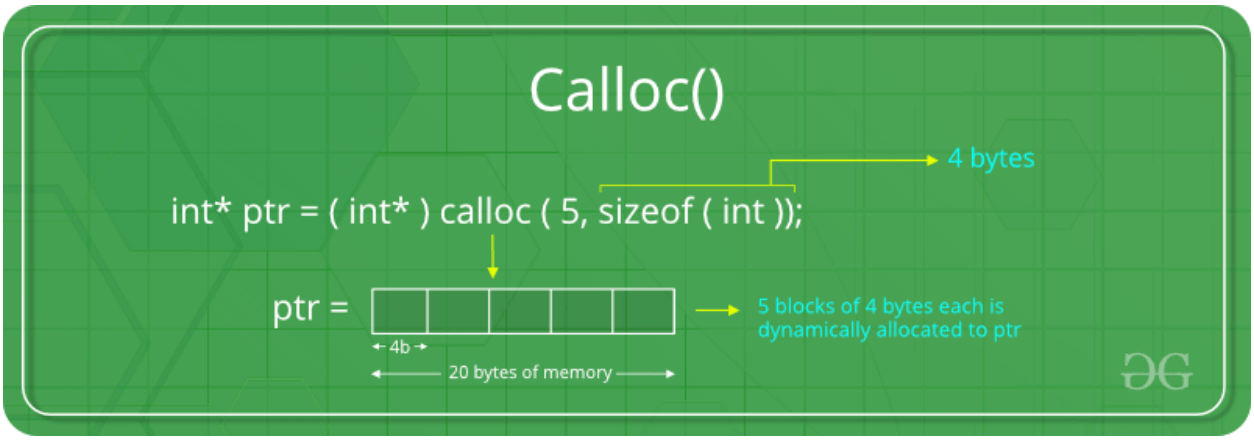
“calloc” or **“contiguous allocation”** method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. It initializes each block with a default value ‘0’. It returns NULL if memory is not sufficient. The syntax of calloc() function is given below:

ptr=(cast-type*)calloc(number, byte-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.




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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int* ptr;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", 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 {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using calloc.\n");

        // Get the elements of the array
        for (i = 0; i < n; ++i) {
            ptr[i] = i + 1;
        }

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

```
    }

    return 0;
}
```

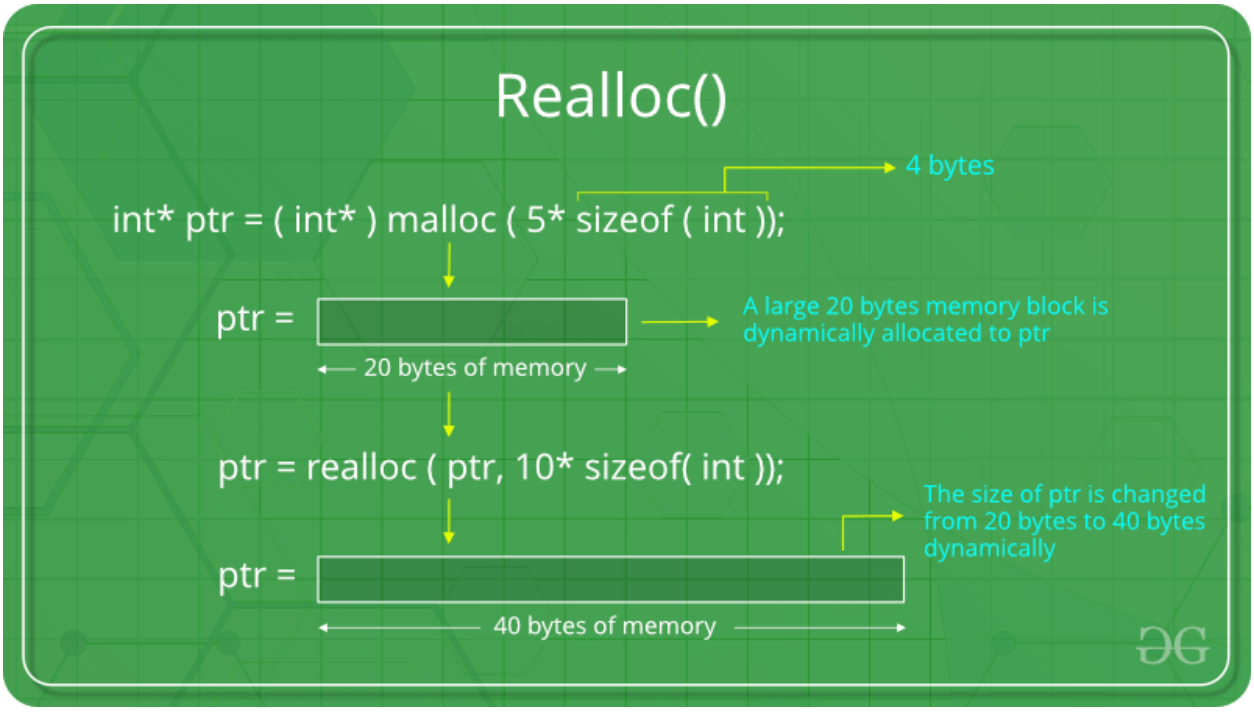
OUTPUT

```
Enter number of elements: 5
Memory successfully allocated using calloc.
The elements of the array are: 1, 2, 3, 4, 5,
```

realloc() function in C

“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**. If space is insufficient, allocation fails and returns a NULL pointer. The syntax of realloc() function is:

ptr=realloc(ptr, new-size);



Example:

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

int main()
{
```

```
// This pointer will hold the
// base address of the block created
int* ptr;
int n, i;

// Get the number of elements for the array
n = 5;
printf("Enter number of elements: %d\n", n);

// Dynamically allocate memory using calloc()
ptr = (int*)calloc(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 {

    // Memory has been successfully allocated
    printf("Memory successfully allocated using calloc.\n");

    // Get the elements of the array
    for (i = 0; i < n; ++i) {
        ptr[i] = i + 1;
    }

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

    // Get the new size for the array
    n = 10;
    printf("\n\nEnter the new size of the array: %d\n", n);

    // Dynamically re-allocate memory using realloc()
    ptr = realloc(ptr, n * sizeof(int));

    // Memory has been successfully allocated
    printf("Memory successfully re-allocated using realloc.\n");

    // Get the new elements of the array
    for (i = 5; i < n; ++i) {
        ptr[i] = i + 1;
    }

    // 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);
    }

    return 0;
}
```

OUTPUT

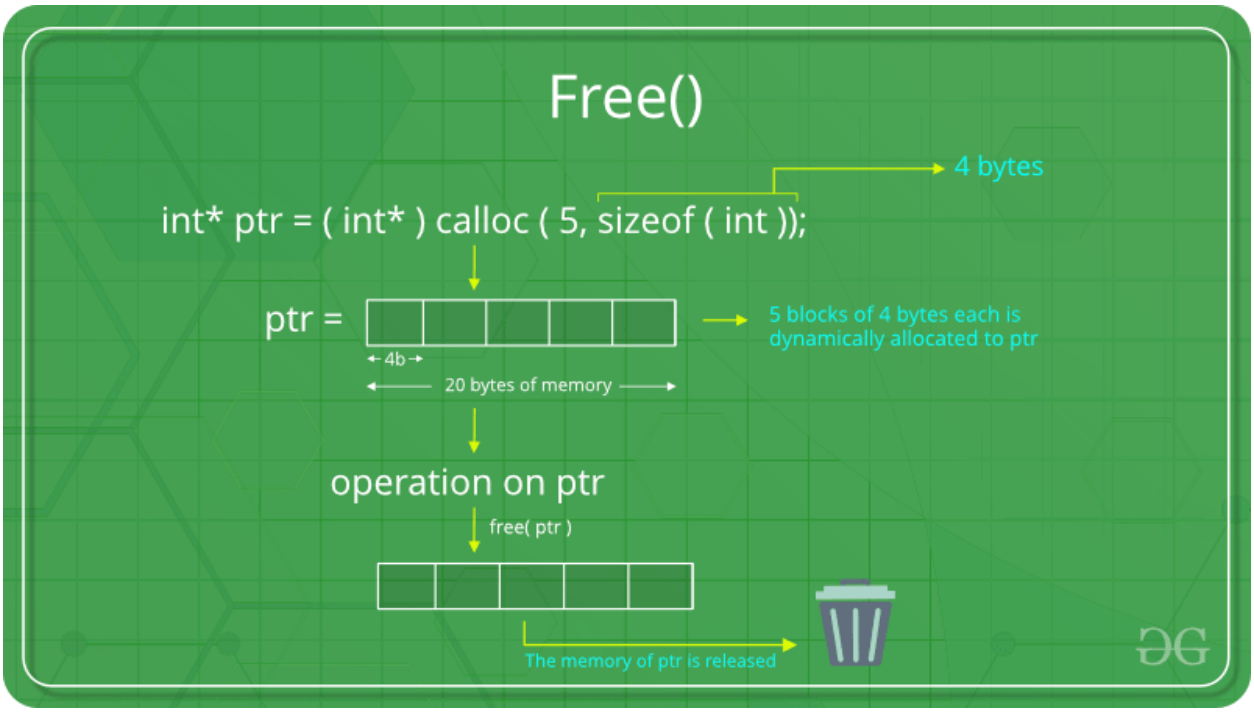
```
Enter number of elements: 5
Memory successfully allocated using calloc.
The elements of the array are: 1, 2, 3, 4, 5,

Enter the new size of the array: 10
Memory successfully re-allocated using realloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

free() function in C

"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. The syntax of free() function is:

```
free(ptr);
```



Example:

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

int main()
{

    // This pointer will hold the
    // base address of the block created
    int *ptr, *ptr1;
    int n, i;

    // Get the number of elements for the array
    n = 5;
    printf("Enter number of elements: %d\n", n);

    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(n * sizeof(int));

    // Dynamically allocate memory using calloc()
    ptr1 = (int*)calloc(n, sizeof(int));

    // Check if the memory has been successfully
    // allocated by malloc or not
    if (ptr == NULL || ptr1 == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {

        // Memory has been successfully allocated
        printf("Memory successfully allocated using malloc.\n");

        // Free the memory
        free(ptr);
        printf("Malloc Memory successfully freed.\n");

        // Memory has been successfully allocated
        printf("\nMemory successfully allocated using calloc.\n");

        // Free the memory
        free(ptr1);
        printf("Calloc Memory successfully freed.\n");
    }

    return 0;
}
```

OUTPUT

```
Enter number of elements: 5
Memory successfully allocated using malloc.
```

```
Malloc Memory successfully freed.

Memory successfully allocated using calloc.
Calloc Memory successfully freed.
```