

## Dynamic Memory Allocation using Array

An array is a collection of a fixed number of values. Once the size of an array is declared, you can't change the size of array.

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

<- Array Indices

**Array Length = 9**

**First Index = 0**

**Last Index = 8**

As it can be seen that the size of above array is 9 but you can't change the size once the size of an array is declared.

To solve this problem, we can allocate memory manually during run-time so that we can reallocate the size of the array which is known as dynamic memory allocation in C programming.

To allocate memory dynamically following library functions are used in C programming and these library functions are defined in the <stdlib.h> header file.

- i. malloc()
- ii. calloc()
- iii. realloc()
- iv. free()

### C malloc() method

"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)                      [ here ptr is a pointer variable ]


#### For Example:

```
int *ptr;  
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. If space is insufficient, allocation fails and returns a NULL pointer.

# Malloc()

`int* ptr = ( int* ) malloc ( 5* sizeof ( int ));`

ptr = 

← 20 bytes of memory →

4 bytes

A large 20 bytes memory block is dynamically allocated to ptr



## 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;
    }
}

```

### C calloc() method

“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);` [ here ptr is a pointer variable ]

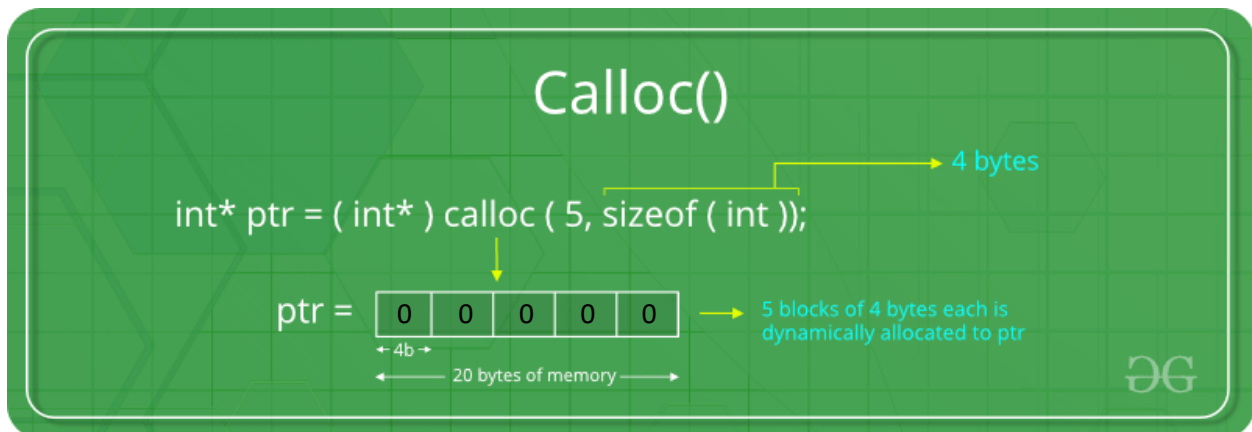
#### For Example:

```

int *ptr;
ptr = (float*) calloc(25, sizeof(float));

```

This statement allocates contiguous space in memory for 25 elements each with the size of the float. If space is insufficient, allocation fails and returns a NULL pointer.



#### 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 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;
}

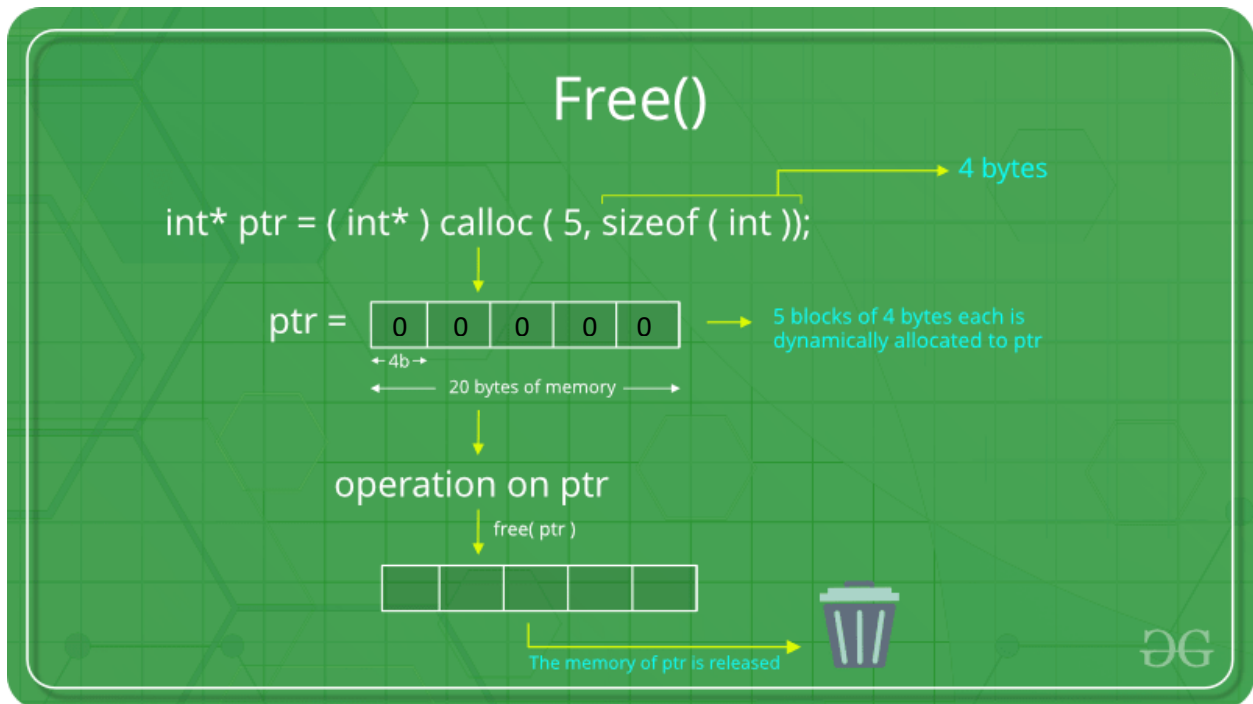
```

### **C free() method**

“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);
```



**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");

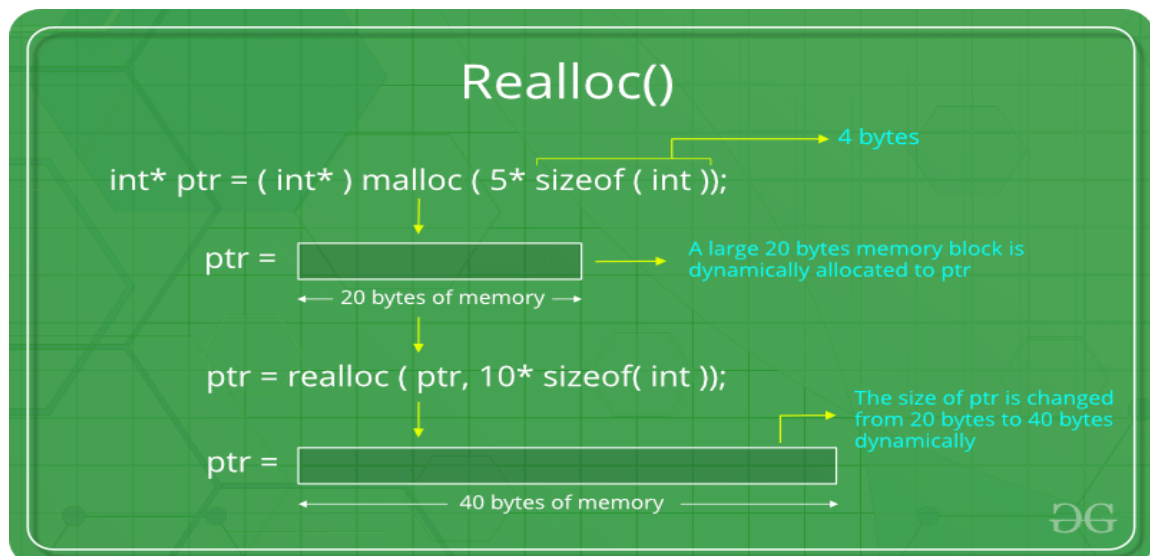
    // Free the memory
    free(ptr1);
    printf("Calloc Memory successfully freed.\n");
}

return 0;
}

```

### C realloc() method

“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'. If space is insufficient, allocation fails and returns a NULL pointer.

**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;
}

```

[source: [www.geeksforgeeks.org](http://www.geeksforgeeks.org)]