**Difference Between Malloc and Calloc**

|  |  |
| --- | --- |
| **malloc** | **calloc** |
| The name malloc stands for *memory allocation*. | The name calloc stands for *contiguous allocation*. |
| void \*malloc(size\_t n) returns a pointer to n bytes of uninitialized storage, or NULL if the request cannot be satisfied. If the space assigned by malloc() is overrun, the results are undefined. | void \*calloc(size\_t n, size\_t size) returns a pointer to enough free space for an array of n objects of the specified size, or NULL if the request cannot be satisfied. The storage is initialized to zero. |
| malloc() takes one argument that is, *number of bytes*. | calloc() take two arguments those are: *number of blocks* and *size of each block*. |
| syntax of malloc():  void \*malloc(size\_t n);  Allocates n bytes of memory. If the allocation succeeds, a void pointer to the allocated memory is returned. Otherwise NULL is returned. | syntax of calloc():  void \*calloc(size\_t n, size\_t size);  Allocates a contiguous block of memory large enough to hold nelements of size bytes each. The allocated region is initialized to zero. |
| malloc is faster than calloc. | 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. |

The malloc() takes a single argument, while calloc() takess two. Second, malloc() does not initialize the memory allocated, while calloc() initializes the allocated memory to ZERO.

Both malloc and calloc are used in C language for dynamic memory allocation they obtain blocks of memory dynamically. Dynamic memory allocation is a unique feature of C language that enables us to create data types and structures of any size and length suitable to our programs.

## Similarities Between Malloc and Calloc

The pointer returned by malloc or callochas the proper alignment for the object in question, but it must be cast into the appropriate type.

Proper alignment means the value of the returned address is guaranteed to be an even multiple of alignment. The value of alignment must be a power of two and must be greater than or equal to the size of a word.

The malloc(), calloc() functions will fail if:

* The physical limits of the system are exceeded by n bytes of memory which cannot be allocated.
* There is not enough memory available to allocate n bytes of memory; but the application could try again later.

**Syntax of Malloc**

void \*malloc(size\_t n);

Allocates n bytes of memory. If the allocation succeeds, a void pointer to the allocated memory is returned. Otherwise NULL is returned.

*/\* Allocate memory for an int. \*/*

int \*ptr = (int\*) [malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(sizeof (int));

if (ptr == NULL)

{

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Could not allocate memory**\n**");

[exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(-1);

}

else

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Memory allocated successfully.**\n**");

**Syntax of Calloc**

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

Allocates a contiguous block of memory large enough to hold n elements of size bytes each. The allocated region is initialized to zero.

*/\* Allocating memory for an array of 10 elements of type int. \*/*

int \*ptr = (int\*) [calloc](http://www.opengroup.org/onlinepubs/009695399/functions/calloc.html)(10 ,sizeof (int));

if (ptr == NULL)

{

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Could not allocate memory**\n**");

[exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(-1);

}

else

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Memory allocated successfully.**\n**");

Dynamic Memory Allocation in C using malloc(), calloc(), free() and realloc()

Since C is a structured language, it has some fixed rules for programming. One of it includes changing the size of an array. An array is collection of items stored at continuous memory locations.  
[](https://www.geeksforgeeks.org/wp-content/uploads/gq/2015/05/Arrays.png)

As it can be seen that the length (size) of the array above made is 9. But what if there is a requirement to change this length (size). For Example, if there is situation where only 5 elements are needed to be entered in this array. In this case the remaining 4 indices are just wasting memory in this array. So there is a requirement to lessen the length (size) of the array from 9 to 5.

Take another situation. In this there is an array of 9 elements with all 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 9 to 12.

These are called **Dynamic Memory Allocation**. 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()
5. **malloc()**

**“malloc”** or **“memory allocation”** method 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)**

For Example:

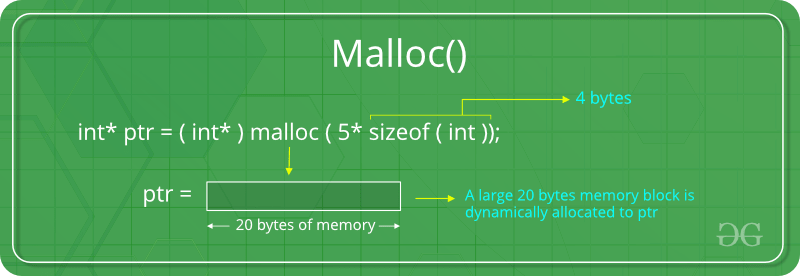
**ptr = (int\*) malloc(100 \* sizeof(int));**

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.



If the 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**,** sum **=** 0**;**    // 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,

1. **calloc()**

**“calloc”** or **“contiguous allocation”** method 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:**

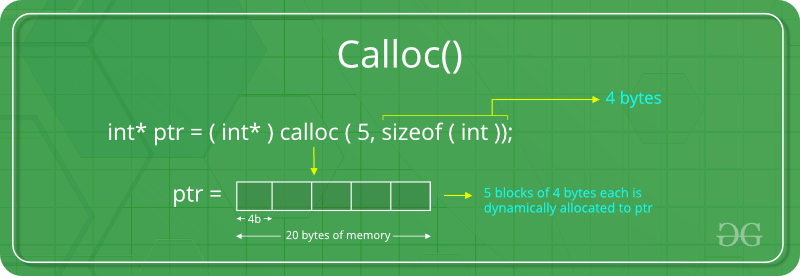
**ptr = (cast-type\*)calloc(n, element-size);**

For Example:

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

This statement allocates contiguous space in memory

for 25 elements each with the size of float.



If the 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**,** sum **=** 0**;**    // 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**]);**  **}**  **}**    **return** 0**;**  **}** |

**Output:**

Enter number of elements: 5

Memory successfully allocated using calloc.

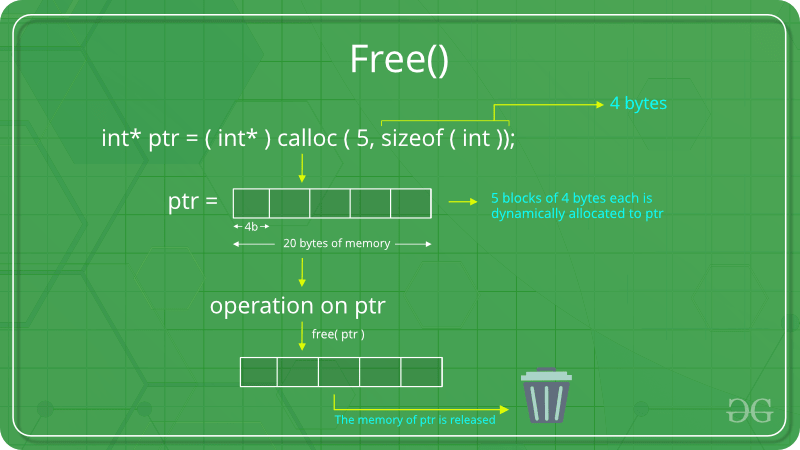
The elements of the array are: 1, 2, 3, 4, 5,

1. **free()**

**“free”** method is used to dynamically **de-allocate** the memory. The memory allocated using functions malloc() and calloc() are 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**,** sum **=** 0**;**    // 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.

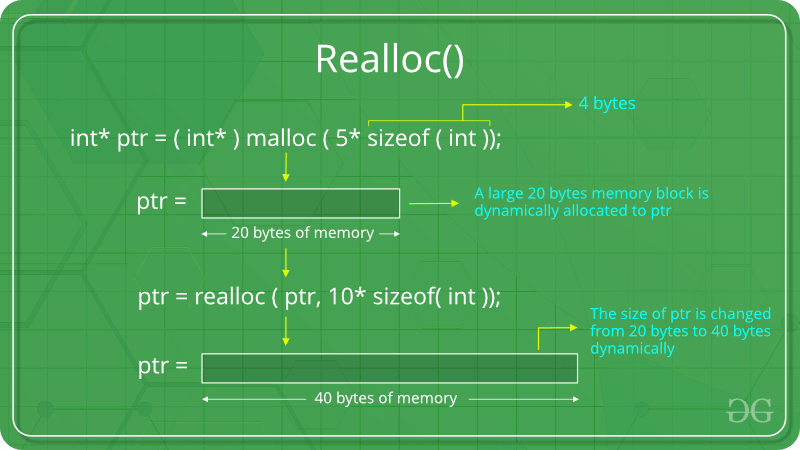
1. **realloc()**

**“realloc”** or **“re-allocation”** method 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 the 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**,** sum **=** 0**;**    // 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,