**Dynamic Memory Allocation in C: malloc(), calloc() Functions**

|  |  |
| --- | --- |
| **static memory allocation** | **dynamic memory allocation** |
| memory is allocated at compile time. | memory is allocated at run time. |
| memory can't be increased while executing program. | memory can be increased while executing program. |
| used in array. | used in linked list. |

## How Memory Management in C works?

When you 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 (according to the platform) when it is declared. We can verify this information using the **sizeof**operator as shown in below example

#include <stdio.h>

int main()

{

float x;

printf("The size of float is %d bytes", sizeof(x));

return 0;

}

The output will be:

The size of float is 4 bytes

Also, an array with a specified size is allocated in contiguous blocks of memory, each block has the size for one element:

#include <stdio.h>

int main()

{

float arr[10];

printf("The size of the float array with 10 element is %d", sizeof(arr)); return 0;

}

The result is:

The size of the float array with 10 element is 40

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 in C 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 in C

**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 you 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 C Dynamic Memory Allocation uses the heap.

The <stdlib.h> library has functions responsible for Dynamic Memory Management.

|  |  |
| --- | --- |
| **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. |

## C malloc() Function

The C malloc() function stands for memory allocation. It is a function which is used to allocate a block of memory dynamically. It reserves memory space of specified size and returns the null pointer pointing to the memory location. The pointer returned is usually of type void. It means that we can assign C malloc() function to any pointer.

Syntax of malloc() Function:

ptr = (cast\_type \*) malloc (byte\_size);

Here,

* ptr is a pointer of cast\_type.
* The C malloc() function returns a pointer to the allocated memory of byte\_size.

Example of malloc():

Example: ptr = (int \*) malloc (50)

When this statement is successfully executed, a memory space of 50 bytes is reserved. The address of the first byte of reserved space is assigned to the pointer ptr of type int.

Consider another example:

#include <stdlib.h>

int main()

{

int \*ptr;

ptr = malloc(15 \* sizeof(\*ptr)); /\* a block of 15 integers \*/

if (ptr != NULL)

{

\*(ptr + 5) = 480; /\* assign 480 to sixth integer \*/

printf("Value of the 6th integer is %d",\*(ptr + 5));

}

}

Output:

Value of the 6th integer is 480

Malloc() function can also be used with the character data type as well as complex data types such as structures.

## The free() Function

The memory for variables is automatically deallocated at compile time. In dynamic memory allocation, you have to deallocate memory explicitly. If not done, you may encounter out of memory error.

**The free()** function is called to release/deallocate memory in C. By freeing memory in your program, you make more available for use later.

For example:

#include <stdio.h>

int main()

{

int\* ptr = malloc(10 \* sizeof(\*ptr));

if (ptr != NULL)

{

\*(ptr + 2) = 50;

printf("Value of the 2nd integer is %d",\*(ptr + 2));

}

free(ptr);

}

Output

Value of the 2nd integer is 50

## C calloc() Function

The C calloc() function stands for contiguous allocation. This function is used to allocate multiple blocks of memory. It is a dynamic memory allocation function which is used to allocate the memory to complex data structures such as arrays and structures.

Malloc() function is used to allocate a single block of memory space while the calloc() in C is used to allocate multiple blocks of memory space. Each block allocated by the calloc() function is of the same size.

Syntax of calloc() Function:

ptr = (cast\_type \*) calloc (n, size);

* The above statement is used to allocate n memory blocks of the same size.
* After the memory space is allocated, then all the bytes are initialized to zero.
* The pointer which is currently at the first byte of the allocated memory space is returned.

Whenever there is an error allocating memory space such as the shortage of memory, then a null pointer is returned.

Example of calloc():

The program below calculates the sum of an arithmetic sequence.

#include <stdio.h>

int main()

{

int i, \* ptr, sum = 0;

ptr = calloc(10, sizeof(int));

if (ptr == NULL)

{

printf("Error! memory not allocated.");

exit(0);

}

printf("Building and calculating the sequence sum of the first 10 terms \ n ");

for (i = 0; i < 10; ++i)

{

\* (ptr + i) = i;

sum += \* (ptr + i);

}

printf("Sum = %d", sum);

free(ptr);

return 0;

}

**Result:**

Building and calculating the sequence sum of the first 10 terms

Sum = 45

## calloc() vs. malloc(): Key Differences

Following is the key difference between [malloc() Vs calloc()](https://www.guru99.com/difference-between-malloc-and-calloc.html) in C:

The calloc() function is generally more suitable and efficient than that of the malloc() function. While both the functions are used to allocate memory space, calloc() can allocate multiple blocks at a single time. You don’t have to request for a memory block every time. The calloc() function is used in complex data structures which require larger memory space.

The memory block allocated by a calloc() in C is always initialized to zero while in function malloc() in C, it always contains a garbage value.

## C realloc() Function

Using the C **realloc()** function, you can add more memory size to already allocated memory. It expands the current block while leaving the original content as it is. realloc() in C stands for reallocation of memory.

realloc() can also be used to reduce the size of the previously allocated memory.

Syntax of realloc() Function:

ptr = realloc (ptr,newsize);

The above statement allocates a new memory space with a specified size in the variable newsize. After executing the function, the pointer will be returned to the first byte of the memory block. The new size can be larger or smaller than the previous memory. We cannot be sure that if the newly allocated block will point to the same location as that of the previous memory block. This function will copy all the previous data in the new region. It makes sure that data will remain safe.

Example of realloc():

#include <stdio.h>

int main ()

{

char \*ptr;

ptr = (char \*) malloc(10);

strcpy(ptr, "Programming");

printf(" %s, Address = %u\n", ptr, ptr);

ptr = (char \*) realloc(ptr, 20); //ptr is reallocated with new size

strcat(ptr, " In 'C'");

printf(" %s, Address = %u\n", ptr, ptr);

free(ptr);

return 0;

}

Whenever the realloc() in C results in an unsuccessful operation, it returns a null pointer, and the previous data is also freed.

1. #include<stdio.h>
2. #include<stdlib.h>
3. **int** main(){
4. **int** n,i,\*ptr,sum=0;
5. printf("Enter number of elements: ");
6. scanf("%d",&n);
7. ptr=(**int**\*)malloc(n\***sizeof**(**int**));  //memory allocated using malloc
8. **if**(ptr==NULL)
9. {
10. printf("Sorry! unable to allocate memory");
11. exit(0);
12. }
13. printf("Enter elements of array: ");
14. **for**(i=0;i<n;++i)
15. {
16. scanf("%d",ptr+i);
17. sum+=\*(ptr+i);
18. }
19. printf("Sum=%d",sum);
20. free(ptr);
21. **return** 0;
22. }

**Output**

Enter elements of array: 3

Enter elements of array: 10

10

10

Sum=30

1. #include<stdio.h>
2. #include<stdlib.h>
3. **int** main(){
4. **int** n,i,\*ptr,sum=0;
5. printf("Enter number of elements: ");
6. scanf("%d",&n);
7. ptr=(**int**\*)calloc(n,**sizeof**(**int**));  //memory allocated using calloc
8. **if**(ptr==NULL)
9. {
10. printf("Sorry! unable to allocate memory");
11. exit(0);
12. }
13. printf("Enter elements of array: ");
14. **for**(i=0;i<n;++i)
15. {
16. scanf("%d",ptr+i);
17. sum+=\*(ptr+i);
18. }
19. printf("Sum=%d",sum);
20. free(ptr);
21. **return** 0;
22. }

**Output**

Enter elements of array: 3

Enter elements of array: 10

10

10

Sum=30

## Dynamic Arrays in C

A Dynamic array in C allows the number of elements to grow as needed. C Dynamic array are widely used in Computer science algorithms.

In the following program, we have created and resized a Dynamic array in C

#include <stdio.h>

int main()

{

int \* arr\_dynamic = NULL;

int elements = 2, i;

arr\_dynamic = calloc(elements, sizeof(int)); //Array with 2 integer blocks

for (i = 0; i < elements; i++)

arr\_dynamic[i] = i;

for (i = 0; i < elements; i++)

printf("arr\_dynamic[%d]=%d\n", i, arr\_dynamic[i]);

elements = 4;

arr\_dynamic = realloc(arr\_dynamic, elements \* sizeof(int)); //reallocate 4 elements

printf("After realloc\n");

for (i = 2; i < elements; i++)

arr\_dynamic[i] = i;

for (i = 0; i < elements; i++)

printf("arr\_dynamic[%d]=%d\n", i, arr\_dynamic[i]);

free(arr\_dynamic);

}

Result of C Dynamic array program at the screen:

arr\_dynamic[0]=0

arr\_dynamic[1]=1

After realloc

arr\_dynamic[0]=0

arr\_dynamic[1]=1

arr\_dynamic[2]=2

arr\_dynamic[3]=3

## Summary

* We can dynamically manage memory by creating memory blocks as needed in the heap
* In C Dynamic Memory Allocation, memory is allocated at a run time.
* Dynamic memory allocation permits to manipulate strings and arrays whose size is flexible and can be changed anytime in your program.
* It is required when you have no idea how much memory a particular structure is going to occupy.
* Malloc() in C is a dynamic memory allocation function which stands for memory allocation that blocks of memory with the specific size initialized to a garbage value
* Calloc() in C is a contiguous memory allocation function that allocates multiple memory blocks at a time initialized to 0
* Realloc() in C is used to reallocate memory according to the specified size.
* Free() function is used to clear the dynamically allocated memory.