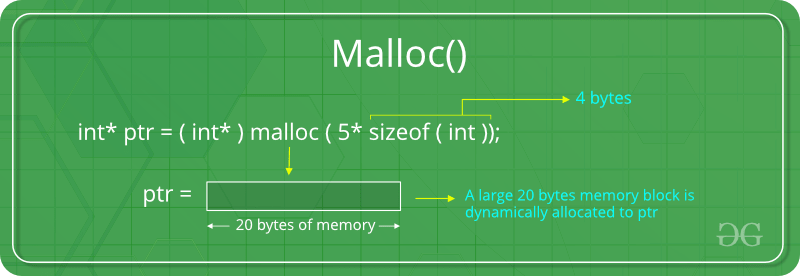
**IMPORTANT:**

There are two **major differences between malloc** and **calloc** in C programming language:

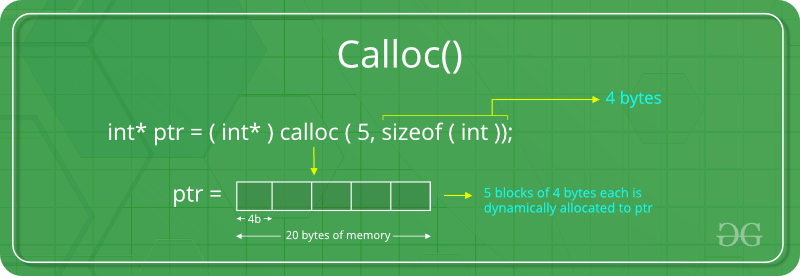
1.First, **in the** number of arguments. The **malloc**() takes a single argument, while **calloc**() takess two.

2.Second, **malloc**() does not initialize the allocated memory, while **calloc**() initializes the allocated memory to ZERO.

**3.“malloc()** “method allocate a **single large** block of memory with the **specified size**. Typecasting is necessary.



**“calloc()”** or **“contiguous allocation”** method allocate the **specified number of blocks** of memory of the specified type. Typecasting is necessary.



**4. malloc** **is fast** comparing to **calloc**. Because calloc allocates memory and initialize memory area with **ZERO**.

Actually the answer is depends on your situation . If you need high performance , use *malloc* . if you more concern on variable initialization , use *calloc*

**3. Print interger/character/string value.**

printf("Print Integer value %d",\*(p)) //

printf("Print chracter value %s",p) //

printf("Print string value %s",p) //

4. Memory Allocation

Ptr = (int\*) malloc(100 \* size(int)); // it also a single argument ( 100\*int)

Considering the size of int is 8 bytes, this statement allocates 800 bytes of memory. And, the pointer *ptr* holds the address of the first byte in the allocated memory.

When we entered first element ptr stored it into first allocated byte,next when we entered second element ptr automatically incremented and store it into next byte.

However, if the space is insufficient, allocation fails and returns a NULL pointer.

0 1 2 3 4 5 6 . . . . 99

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

Ptr

Program:

{

int n,i,\*p;

printf("Enter number of elements:\n");

scanf("%d",&n);

p=(int\*)malloc(n \* sizeof(int));

{

printf("Enter elements of array:\n");

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

{

scanf("%d",&\*(p+i)); //scanf("%d",(p+i));

}

printf("Elements of array are\n");

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

{

printf("%d\n",\*(p+i));

}

}

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | 13 | 14 | 15 |

ptr

5.Difference:

A)The name "calloc" stands for contiguous allocation.

B)The malloc() function allocates a single block of memory.

C)Whereas, calloc() allocates multiple blocks of memory and **initializes** them to **zero**.

6.Relloc()

ptr = realloc(ptr, x); //Here, *ptr* is reallocated with new size *x*.

**4. Consider following Program for printing Character value only**

**char \*p = calloc(100, 1);**

**p = "welcome";**

**printf("%c\n", p); // use %s to print welcome. Otherwise it print T**

**5. Array:**

**a. Storing value in Array**

**Printf("Enter elements of array:\n");**

**for(i=0;i<n;++i)**

**{**

**scanf("%d",&\*(p+i));**

**OR**

**scanf(“%d”,(p+i));**

**}**

**b.Print Elements**

**printf("Elements of array are\n");**

**for(i=0;i<n;i++)**

**{**

**printf("%d\n",\*(p+i)); //use \*(p+i)**

**}**

**Why use Dynamic Memory Allocation?**

1. **Dynamic Memory** Allocation means to **Allocate Memory at run-time**i.e at the time of Program execution.
2. In C **malloc and calloc** functions are used to allocate memory at Run-time
3. It **returns pointer to block** of n bytes of memory allocated during run-time.
4. If the Process fails i.e if it is unable to allocate memory then **returns NULL**
5. **Note :**Typecasting is necessary.
6. Calloc is more efficient than malloc because it allocates memory in 1 clock cycle

Dynamic memory allocation in c programming will allow program to allocate/release memory at run time

**a.** At run time you want to accept more numbers from user than size of array.

b. At run time you want to accept very few numbers than size of array.

Example:

In this program , we have not declared two variables at compile time. We are going to accept values at run time by allocating memory at run time.

#include<stdio.h>

#include<stdlib.h>

**int** main()

{

**int** \*ptr,i;

// Allocate memory Equivalent to 1 integer

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

**for**(i=0;i<3;i++)

{

printf("Enter %d number : ",i+1);

scanf("%d",(ptr+i));

}

**for**(i=0;i<3;i++)

{

printf("%d\n",\*(p+i));

}

**return**(0);

}

In the above program we have considered that , Memory will be available at any moment of time. Now Suppose Memory is unavailable at run time then we are unable to allocate memory. Executing dynamic memory allocation function would cause System Crash.

**if**(ptr==NULL)

{

printf("\nError in Allocating Memory");

exit(0);

}

Suppose memory is unavailable at run time then it will return NULL. We are checking the value of pointer with NULL. If legal address is provided by calloc or malloc then we can consider that memory is allocated successfully , however if these memory allocation functions return NULL then we can say that some error occurred while allocating memory.

1. calloc() initializes the allocated memory to **zero** value whereas malloc **doesn't**.

For Example:-

#include <stdio.h>

#include <stdlib.h>

int main()

{

int n,i,\*p;

printf("Enter number of elements: ");

scanf("%d",&n);

p=(int\*)calloc(n, sizeof(int));

{

printf("Elements of array are\n");

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

{

printf("%d\n",\*(p+i));

}

}

return 0;

}

Output: Enter number of elements: 2

Elements of array are

0

0

***calloc*** *is used to allocate memory to mostly* ***arrays*** *and* ***structures***

3. A condition where in memory is reserved dynamically but not accessible to any of the programs is called “**Memory Leak**”. Memory leak and dangling pointers are not inter dependent. Hence, when memory leak occurs, it is not mandatory that a dangling pointer arises

4. #include**<stdlib.h>** for dynamic operation. Pointer must used for it.

5.integer pointer point integer type of variable and float/character pointer point float/character type of variables.

6. a=(int\*)malloc(sizeof(int))

means you are allocating space off the heap to store a pointer to an **int** . This returns a value you should cast to **int** \* . (A pointer to an **int** .)

7.Program

#include<stdio.h>

#include<stdlib.h>

void main()

{

char \*p = calloc(100, 1);

p = "welcome";

printf("%s**\n**", p);

}

Output: welcome(There is no error in the above code. The format specifier being %s, address is not returned. Hence, welcome is the output.)

8. Local variables are stored on the stack. Static variables are stored in the permanent storage area.

9.**Relloc:**

Program:

#include<stdio.h>

#include<stdlib.h>

main()

{

int \*p,i,a,b;

printf("Enter size of array");

scanf("%d",&a);

p=(int\*)malloc(sizeof(int)); // p=(int\*)malloc(a \*sizeof(int));//We can used ‘a’.

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

printf("%d\n",i);

printf("Enter new size of array");

scanf("%d",&b);

realloc(p,b);

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

printf("%d\n",i);

free(p);

}

Output: Enter size of array2

0

1

Enter new size of array3

0

1

2

10. If the space in memory allocated by malloc is not sufficient, then an allocation fails and return

**NULL pointer.**

**11. Dynamic memory allocation** in C/**C++** refers to performing **memory allocation** manually by programmer. **Dynamically allocated memory** is **allocated** on Heap and non-static and local variables get **memory allocated** on Stack.

11.Important:

#include<stdio.h>

#include<stdlib.h>

int main()

{

int \*p;

p = (int \*)malloc(20);

printf("%d**\n**", sizeof(p));

free(p);

return 0;

}

Output:8(64 bit plateform)

**12.Why we used P instead of \*P for getting value pointed by p.**

**Example: 1**

#include<string.h>

int main()

{

char \*p;

p = (char\*)calloc(10,2);

strcpy(p, "HELLO");

printf("%s", p);

free(p);

return 0;

}

Output: HELLO

Example: 2

#include<string.h>

#include <stdlib.h>

#include <string.h>

int main()

{

char \*address;

address = (char\*)malloc(sizeof(char) ); /\* allocating memory dynamically \*/

strcpy( address, "Lee Fort, 11-B Sans Street");

printf("Address: %s\n", address );

return 0;

}

Output: Address: Lee Fort, 11-B Sans Street

**\* Calloc() is used to allocate memory to mostly arrays and structures.**

***/\* Dynamically Allocated Memory using Array \*/***

**Program:1**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main()**

**{**

**int n,i,\*p;**

**printf("Enter number of elements:\n");**

**scanf("%d",&n);**

**p=(int\*)malloc(n \* sizeof(int));**

**{**

**printf("Enter elements of array:\n");**

**for(i=0;i<n;++i)**

**{**

**scanf("%d",&\*(p+i)); OR //scanf(“%d”,(p+i));**

**}**

**printf("Elements of array are\n");**

**for(i=0;i<n;i++)**

**{**

**printf("%d\n",\*(p+i));**

**}**

**printf("size before realeasing memeory %d\n",\*(p));**

**}**

**free(p);**

**printf("size after realeasing memory %d",\*(p));**

**return 0;**

**}**

**Output:**

**Enter number of elements:**

**2**

**Enter elements of array:**

**1**

**2**

**Elements of array are**

**1**

**2**

**size before realeasing memeory 1**

**size after realeasing memory 0**