

Dynamic Memory Allocation :

- ▶ This procedure is referred to as **Dynamic Memory Allocation in C**.
- ▶ Therefore, **C Dynamic Memory Allocation** can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.
- ▶ 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.
 1. malloc()
 2. calloc()
 3. free()
 4. realloc()

1. malloc() :

- ▶ The “**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.
- ▶ It doesn't Initialize memory at execution time so that it has initialized each block with the default garbage value initially.

- ▶ **Syntax of malloc() :**

$p = (\text{cast-type}^*) \text{malloc}(\text{byte-size})$


- ▶ **$p = (\text{int}^*) \text{malloc}(100 * \text{sizeof}(\text{int}));$** or

$p = (\text{int}^*) \text{malloc}(50);$

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

Malloc()

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



ptr =



← 20 bytes of memory →

→ A large 20 bytes memory block is dynamically allocated to ptr



```

#include<stdio.h>
#include<stdlib.h>
//#include<alloc.h>
void main() {
    int *p,i,n;
    clrscr();
    printf("Enter Size of dynamic
    array");
    scanf("%d",&n);
    //malloc
p=(int*)malloc(n*sizeof(int));
    //it will allocate dynamic
    memory using malloc() default
value garbage .

```

```

for(i=0;i<n;i++)
{
    printf("\np[%d]",i);
    scanf("%d",&p[i]);
}
for(i=0;i<n;i++)
{
    printf("\n p[%d]%d",i,p[i]);
}
free(p);
getch();
}

```

2. calloc() :

- ▶ “**calloc**” or “**contiguous allocation**” method in C is used to dynamically allocate the specified number of blocks of memory of the specified type. it is very much similar to malloc() but has two different points and these are:
 1. It initializes each block with a default value ‘0’.
 2. It has two parameters or arguments as compare to malloc().
- ▶ **Syntax of calloc() :**

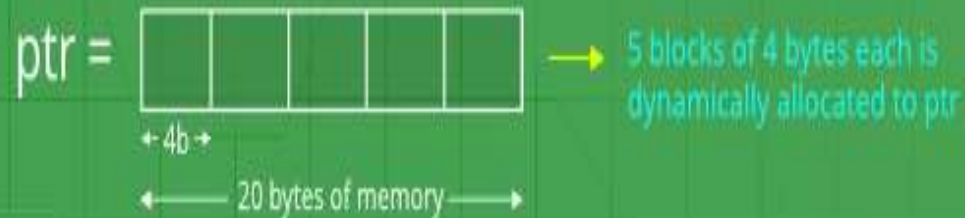
$p = (\text{cast-type}^*)\text{calloc}(n, \text{element-size});$
here, n is the no. of elements and element-size is the size of each element.
- ▶ **$p = (\text{int}^*) \text{calloc}(25, \text{sizeof}(\text{int}));$**

This statement allocates contiguous space in memory for 25 elements each with the size of the int.

Calloc()

```
int* ptr = (int*) calloc ( 5, sizeof ( int ));
```

4 bytes



```
#include<stdio.h>
#include<stdlib.h>
//#include<alloc.h>
void main() {
    int *p,i,n;
    clrscr();
    printf("Enter Size of dynamic
    array");
    scanf("%d",&n);
p=(int*)calloc(n,sizeof(int));
    //it will allocate dynamic
    memory using calloc(), default
value 0 .
```

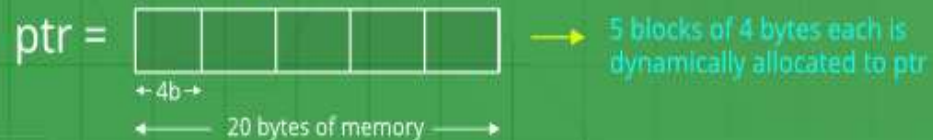
```
for(i=0;i<n;i++)
{
    printf("\np[%d]",i);
    scanf("%d",&p[i]);
}
for(i=0;i<n;i++)
{
    printf("\np[%d]%d",i,p[i]);
}
free(p);
getch();
}
```

3. free() :

- ▶ **“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 of free() :**
 free(p);

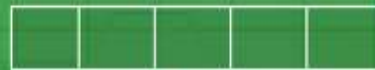
Free()

```
int* ptr = ( int* ) calloc ( 5, sizeof ( int ));
```



operation on ptr

free(ptr)



The memory of ptr is released




4. realloc() :

- ▶ “**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**.
- ▶ re-allocation of memory maintains the already present value and new blocks will be initialized with the default garbage value or 0.
- ▶ **Syntax of realloc() :**
- ▶ `p = realloc(p, newSize);`
where p is reallocated with new size 'newSize'.

Realloc()

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

4 bytes

ptr = 

← 20 bytes of memory →

A large 20 bytes memory block is dynamically allocated to ptr

```
ptr = realloc ( ptr, 10* sizeof( int ));
```

ptr = 

← 40 bytes of memory →

The size of ptr is changed from 20 bytes to 40 bytes dynamically



```

#include<stdio.h>
#include<stdlib.h>
void main() {
    int *p,i,n;
    clrscr();
    printf("Enter Size of dynamic array");
    scanf("%d",&n);
    p=(int*)calloc(n,sizeof(int));
    //it will allocate dynamic memory using
    calloc(), default value 0 .
    for(i=0;i<n;i++)
    {
        printf("\np[%d]",i);
        scanf("%d",&p[i]);
    }
    for(i=0;i<n;i++)
    {
        printf("\np[%d]%d",i,p[i]);
    }
}

```

//realloc re-allocation of memory

```

printf("Enter new Size of dynamic
array");
scanf("%d",&n);
p=(int *)realloc(p,n*sizeof(int));
for(i=0;i<n;i++)
{
    printf("\np[%d]",i);
    scanf("%d",&p[i]);
}
for(i=0;i<n;i++)
{
    printf("\np[%d]%d",i,p[i]);
}
free(p);
getch();
}

```