DYNAMIC MEMORY ALLOCATION (DMA)

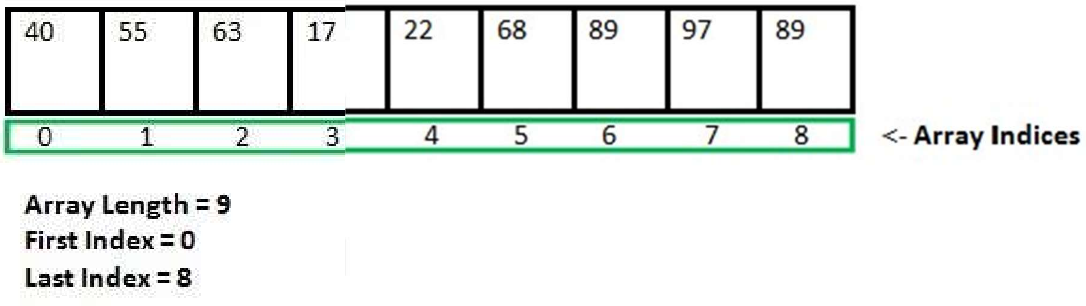
1. STATIC OR COMPILE TIME ALLOCATION

(using array)

2. DYNAMIC OR RUN-TIME ALLOCATION

(using pointor)

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.



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 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 a 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.**

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. They are:

1. **malloc()**
2. **calloc()**
3. **free()**
4. **realloc()**

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. It initializes each block with default garbage value.

Syntax:

ptr = (cast-type*) malloc(byte-size)

eg.

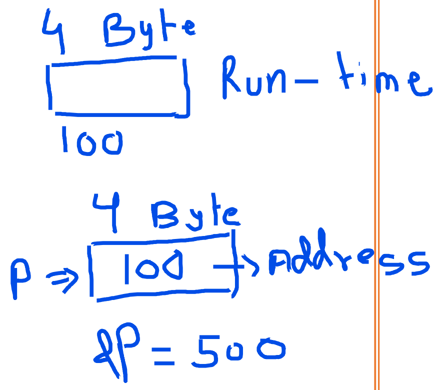
1. for ingeter data

int *p ; OR int* p ; type of p --> int *

p = (int*) malloc (4);

int type void Address

Casting



Since the size of int is 2 bytes, this statement will allocate 2 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

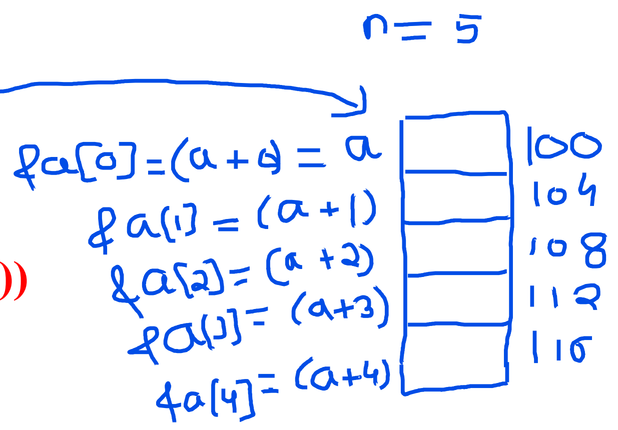
If space is insufficient, allocation fails and returns a NULL pointer.

2. one dim. array

int *a ;

5 × 4

a = (int *) malloc (n * sizeof(int))



&a[i] = (a + i)
a[i] = *(a + i)

a → 100 → Address
&a = 500

Example: // DYNAMIC INPUT AND PRINT N NOS.

```
#include<stdio.h>
#include<malloc.h>


 int    main()
 {
        int  *a , i , n ;

       printf(" ENTER SIZE \n");

       scanf("%d",&n);

       a = (int*) malloc ( n * sizeof(int) );
```

// CREATE N SIZE OF ARRAY IN RUN TIME

```
        printf("ENTER NO \n");


        for( i = 0 ; i < n ; i++)
        {
                scanf("%d",&a[i]);
        }
        printf(" NO  = \n ");

        for( i = 0 ; i < n ; i++ )
         {
                printf("%d\n" , a[i]);
         }
            free(a);
  }
```

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);

e.g free(a); ⟶ a → □ → Address
&a = 500

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'.

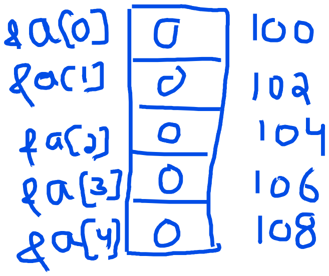
Syntax: **ptr = (cast-type*)calloc(n, element-size);**

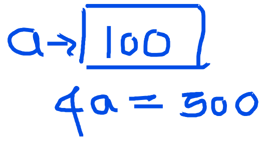
This statement allocates contiguous space in memory for n elements each with the size of the data type.

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

one dim. array

```
int *a;

a = (int*) calloc (n , sizeof(int));

            OR

a = (int*) calloc (n , 2);
```





DIFFERENCE B/W MALLOC AND CALLOC

	malloc	calloc
DEFAULT VALUE	garbage value	zero

```
    // Example of Calloc with N size
#include<stdio.h>
#include<malloc.h>

int  main()
{
        int *a,i,n;

        printf("Enter Size=");

        scanf("%d",&n);

        a = (int*) calloc ( n , sizeof(int) );

        printf("No=\n");

        for(  i = 0 ; i < n ; i++)
        {
                printf("%d\t",a[i]);  // 0    0    0
        }
        free(a);

}
```

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. re-allocation of memory maintains the already present value and new blocks will be initialized with default garbage value.

Syntax: **ptr = realloc(ptr, newSize);**

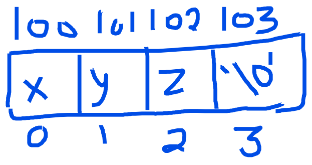
where ptr is reallocated with new size 'newSize'.

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

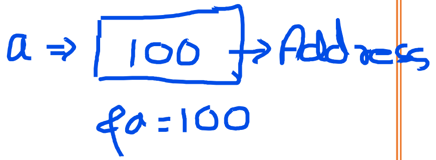
Example: **DYNAMIC INPUT AND PRINT NAME.**

```
#include<stdio.h>
#include<malloc.h>

int  main()
{
        int n
        char *a;

        printf("enter size\n");
        scanf("%d",&n);  //   n = 3
```

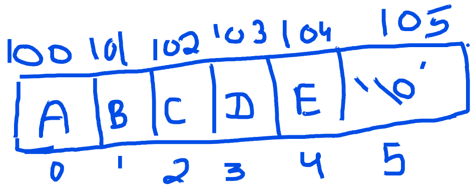
```
    a = (char*) malloc ( (n+1) * sizeof(char));
```



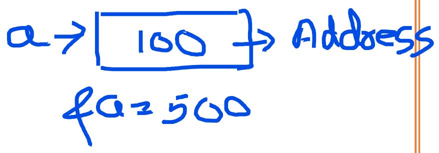
```
    printf("enter name\n");
    scanf("%s",a);  // xyz
```



```
    printf("a=%s\n",a);

    printf("enter new size\n");
    scanf("%d", &n);  // n = 5

    a = (char*) realloc ( a,(n+1));
```



```
    printf("enter name\n");
    scanf("%s",a);  //ABCDE
```



```
    printf("a=%s\n",a);

    free(a);
}
```