* Dynamic Memory Allocation (DMA): (MA)(4)

An array is a collection of a fixed number of values. once the size of the array is declared, you cannot change it.

→ Sometimes the size of the array declared may be insufficient. To solve this issue, you can allocate memory manually during run-time.

This is known as dynamic memory allocation in C programming.

→ To allocate memory dynamically, library functions are `malloc()`, `calloc()`, `realloc()` and `free()` are used. These functions are defined in the `<stdlib.h>` header file.

* **malloc():-** The name "malloc" stands for memory allocation.

The `malloc()` function reserved a block of memory of the specified number of bytes. it return a pointer of `void` which can be casted into pointers of any form.

↳ **Syntax of malloc():**

```
Ptr=(CastType*) malloc(size);
```

example:

```
Ptr=(float*) malloc(100*sizeof(float));
```

* The above statement allocates 400 bytes of memory. It's because the size of `float` is 4 bytes. the pointer `Ptr` holds the address of the first byte in the allocated memory.

Scanned with CamScanner

→ The above expression results in a NULL pointer if the memory cannot be allocated. (MA) (5)

* **Calloc ():** The name "Calloc" stands for Contiguous allocation.

→ The Malloc() function allocates memory and leaves the memory uninitialized. whereas the Calloc() function allocates memory and initilizes all bit to zero.

↳ **Syntax of Calloc():**

```
Ptr = (CastType*) Calloc(n, size);
```

Example:

```
Ptr = (float*) Calloc(25, sizeof(float));
```

* The above statement allocates contiguous space in memory for "25" elements of type float. means that 4 byte * 25 = 100 byte of contigous space.

* **free ():** Dynamically allocated memory created with either Calloc() or Malloc() doesn't get freed on their own. you must explicitly use free() to release the space.

↳ **Syntax:**

```
free (ptr);
```

* This statement frees the space allocates in the memory pointed by Ptr.

Scanned with CamScanner

* **Realloc():-** If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using the **realloc()** function. (MA) (6)

↳ **Syntax:**

```
Ptr=realloc(Ptr, x);
```

Here, **Ptr** is reallocated with a new size **x**.

Example of these function in program:

```
#include<Stdio.h>
#include<stdlib.h>
 void main(){
 int n, i, *p1, *p2, sum1=0, sum2=0;
Printf("Enter the no. of element");
 Scanf("%d", &n);
p1= (int*)malloc(n*sizeof(int));
p2= (int*)Calloc(n, sizeof(int));
 if(p1==NULL){
 Printf("memory not allocated");
    exit();
 }
 if(p2==NULL){
Printf("Memory not allocated");
   exit();
 }
Printf("Enter elements");
for(i=0; i<n; i++){
 Scanf("%d", p1+i);
  sum1=sum1+*(p1+i);
 Scanf("second %d", p2+i);
   sum2=sum2+*(p2+i);
 }
Printf("sum1=%d", sum1);
Printf("sum2=%d", sum2);
  free(p1);
  free(p2);
}
```

Program for realloc()

```
#include<stdio.h>
#include<stdlib.h>
 void main(){
  int *ptr, i, n1, n2;
 Printf("Enter size");
  Scanf("%d", &n1);
 Ptr= (int*)malloc(n1*sizeof(int));
Printf("Address of memory location");
 for(i=0; i<n1; i++)
  Printf("%u", ptr+i);
Printf("Enter new size");
 Scanf("%d", &n2);
Ptr=realloc(Ptr, n2*sizeof(int));
Printf("New Address after relocation");
 for(i=0; i<n2; i++)
  Printf("%u", ptr+i);
  free(ptr);
}
```

Output:

```
Enter Size: 2
Address of memory location
  123123, 123125
Enter New size: 3
123123, 123125, 123127
```

Scanned with CamScanner