malloc() → allocates single block of continuous memory on the heap at runtime.

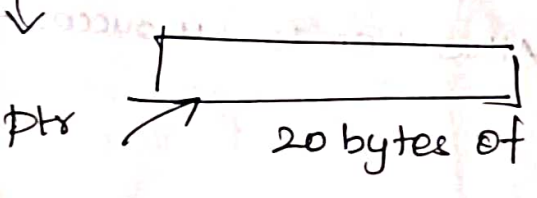
uninitialized → so garbage values is stored.

→ returns void pointer to the allocated memory that needs to be converted to the pointer of req. type.

If allocation fails → it returns the NULL pointer.

ex: `int *ptr = (int *)malloc(20);`

If no memory available, the malloc will fail, hence check for failure by comparing to NULL.



2) calloc() → function.

callocc() → stands for contiguous allocation.,

→ similar to malloc() but it initializes the allocated memory address to zero.

→ used when you need memory with default zero values.

→ returns void pointer, converted to req. type

If allocation fails → returns Null pointer.

Syntax

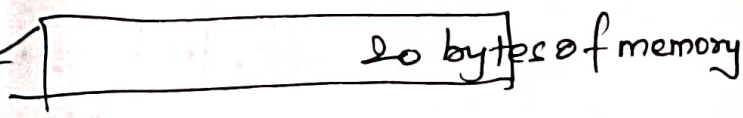
```
Void *calloc(n, size);
```

n → num. of elements to be allocated

size → byte size of each element.

example:-

```
main()
{
    int N, *a, i, S=0;
    Printf("Enter the no. of elements");
    scanf("%d", &N);
    a= (int *) calloc(N, sizeof(int));
  if(a==NULL)
  {
    Printf("alloc is unsuccessful");
  }
  Printf("enter the array elements
                  1 by 1");
  for(i=0; i<N; i++)
  {
    scanf("%d", &a[i]);
      S= S+a[i];
  }
 Printf("%d", s);
}
```



ex: int *ptr= (int *) calloc (5, sizeof (int));

3) realloc() → re-allocation

It is used to resize a previously allocated memory block.

→ allows you to change the size of an existing memory allocation without needing to free the old memory & allocate a new block. (Increase | Decrease the allocated memory)

Syntax:-

```
Void *realloc (*ptr, size);
```

*ptr → pointer to the memory block previously allocated by calloc | malloc | realloc.

→ if NULL, a new block is allocated.

size → new size of the memory block, if "0", ptr points to an existing block of memory; memory block pointed by ptr is deallocated and NULL pointer is returned.

reallocated size

→ returns void pointer to the newly allocated memory or NULL if reallocation fails.

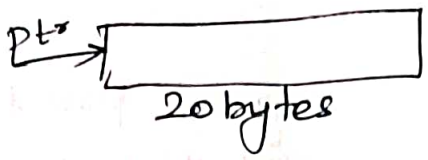
if, fails → original memory block remains unchanged.

ex)

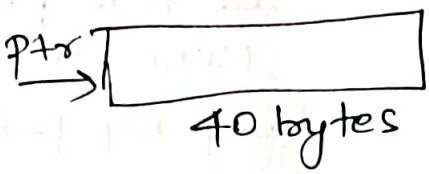
```
int *ptr = (int *) malloc (5 * sizeof (int));
```

Resize,

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



malloc (5 * sizeof(int));



realloc (ptr, 10 * sizeof(int))

Shrink,

```
Ptr = (int *) realloc (ptr, 5 * sizeof (int));
```

4) free

→ memory allocated using functions malloc() & calloc() is not de-allocated on their own.

free() → is used to release dynamically allocated memory back to the OS.

It is necessary to avoid memory leaks.

syntax

```
void free(void *ptr);
```

ptr → pointer to block of mem previously allocated.

ex)

```
int *ptr = (int *)calloc (5, sizeof(int));
// Do some operations.
```

```
for( int i=0; i<5; i++)
    Printf ("%d", ptr[i]);

//free
    free(ptr);

set
    ptr = NULL;
```

→ good practice to set the pointer to NULL, to avoid "dangling pointer".

5) Structures

Arrays, strings → handle similar data

To handle dissimilar data } → structures

eg) 'book' contains collection of items like title, author, publisher, no. of pages, date of pub etc.

Structure — A collection of variables of different datatypes.

It is a user defined datatype.

that holds collection of elements of different type, under a single name.

Array	Structures
same type of ele.	different type of ele,
homogenous data	contains heterogenous data.