# Ch- Dynamic Memory Allocation in C

↳ Static : int arr [3] = {1, 2, 3};
  Memory (Stack):
   [1000] -> 1
   [1004] -> 2
   [1008] -> 3

↳ Dynamic : int *ptr = malloc (3*sizeof (int));
  Memory (Heap):
   [2000] -> [uninitialized]
   [2004] -> [uninitialized]
   [2008] -> [uninitialized]

•> Dynamic memory allocation lets us request memory at runtime, not compile time — flexible sizes!!

- Static - fixed size ( eg. int arr[10])
- Dynamic - size is decided during execution (e.g. user input)

ex- write a program to declare a static array of 5 ints & print size using "sizeof"

```
#include <stdio.h>
   int main() {
        int arr[5] = {1,2,3,4,5};
         printf ("size: %zu \n", sizeof(arr));
         return 0;
   }
```

## 2.> Malloc

```
int *ptr = malloc ( 3* sizeof (int));
Memory (Heap);
[3000] -> [garbage];
[3004] -> [garbage];
[3008] -> [garbage];
ptr = 3000
```

↳ What's Malloc → allocates a block of memory in bytes. Returns a (*void - uninitialized)

↳ ex- allocate memory for 4 floats, assign values, & print them.

```
#include <Stdio.h>
#include <Stdlib.h>
int main () {
    float *ptr = (float *) malloc (4 * sizeof (float));
    ptr[0] = 1.1; Ptr[1] = 2.2; Ptr[2] = 3.3; ptr[3]=4.4;
    for( int i = 0; i<4; i++ ) {
        printf(" %.0lf \n", Ptr[i]);
    }
    free (ptr);
    return 0;
}
```

## 3.) Calloc

```
int *ptr =  calloc ( 3, sizeof (int));
```

Memory (Heap):

[4000] -> 0
[4004] -> 0
[4008] -> 0

ptr = 4000

- it's like calloc. But initializes ~~to~~ all bytes to 0.
- Syntax: calloc (num_elements, size-per-element).
- ex- use "calloc" for 5 chars, assign 'A to E' and print as a string.

```
#include <stdio.h>
#include <stdlib.h>
 int main() {
        char *ptr = (char*) calloc (5, sizeof (char));
         for (int i = 0; i < 5; i++) {
              ptr[i] = 'A' + i;
           }
        printf ("%s \n", ptr);
         free (ptr);
         return 0;
       }
```

⇒ Malloc leaves garbage, calloc doesn't know what you need.

4.> Realloc

```
int *ptr = malloc(2 * sizeof(int));
  [5000] --> 1
  [5004] --> 2
ptr = realloc (ptr, 4 * sizeof(int));
[6000] -> 1            //New block (if moved)
[6004] -> 2
[6008] ->     [uninitialized]
[6012] ->     [uninitialized]
    ptr = 6000
```

.> It resizes previously allocated block. Keeps data if possible.

.> grow and shrink dynamically.

```
int *ptr = malloc (2*sizeof (int));
   ptr[0] = 1;  ptr[1] = 2;
ptr = realloc (ptr, 4* sizeof( int));
   ptr[2] = 3;  ptr[3] = 4;
```

5.> free

.> Releases allocated memory back to the system.
.> syntax: free (ptr);

ex- .> allocate for 3 ints, set values, print, free & try printing again.

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

GROOT
ACADEMY

```
int main (){
    int *ptr= malloc (3* sizeof (int));
     ptr[0]= 5;  ptr[1] = 10; ptr[2] = 15;
      printf("%d \n", ptr[0]);
     free (ptr);
    retuen 0;
}
```