← بسم الله الرحمن الرحيم

Dynamic memory allocation:-

↳ Heap → Controlled by Programmer

① malloc() → reserve block of memory of specific size

↳ Returns the address of the allocated memory (void *)

```
int n,
int * Ptr;
Ptr = (int *) malloc(n * sizeof(int));
```

$\underbrace{\text{(int *)}}_{\text{Type casting}}$ — $\underbrace{n}_{\text{you can dynamically take (n) value from user}}$

② free(n) Ptr of allocated memory

↳ unlike variables, arrays ; the allocated memory using malloc is not freed up by itself

Note : you should use Null safety with malloc ()
to avoid errors if the allocation failed

```
int * Ptr = (int *) malloc( n * sizeof(int));
if ( Ptr == NULL) return 1;
else    ----

    ----
    free(Ptr);
    return 0;
```

③ realloc( ) → Return

↳ to increase the memory you have Previously allocated using malloc( )

Ex: You allocated 8 bytes at first and then you need 2 more bytes

(void*) realloc (Ptr, $\overset{(n+2)}{n}$ * sizeof int);

Ptr ↳ adrress of Previously allocated memory

n * sizeof int ↳ New size

→ Return ~~new~~ address of the resized memory

↳ * Same as the Previous address
↳ if there is enough Contiguous space for the new allocated space

* New address
↳ if there is No enough Contiguous space and it also free. the old adress and Copy its data to a new adress

4 Calloc ( ) - Contiguous allocation

↳ - allocate memory for array elements
 - initialize them to Zero / \0 / Null
 - Return base address

Calloc (number of elements, size of each element)

Ex int *arr = Calloc (5, sizeof(int));

↗ Array of 5 integers each of value 0

Preprocessors & macros

include <stdio.h> → executed before compiling

include : add external header files → stdio.h : Printf(), Scanf()
→ math.h : sqrt(), Pow()
→ ...

define : define macros "Constants"
Ex: # define PI 3.14

Faster than variables Consts

You can also define a function macro
define circleArea(r) (PI * r * r)

Faster than Function calls

No Type checking

include guards

↳ to avoid double inclusion
↳ including the Same header file more than once

Ex

Course.h

```
#ifndef COURSE_H  -> if Course_h not defined
#define Course_H
#include "Student.h"

typedef struct {
    student Students [10]
    char name [10];
} Course;

#endif -> end if
```

Student.h

```
#ifndef STUDENT_H
#define STUDENT_H
typedef struct{
    ____
} student
endif
```

macro names Should be unique.

You Can use #Pragma once instead

```
#Pragma once
typedef struct{
    ____
} student;
```