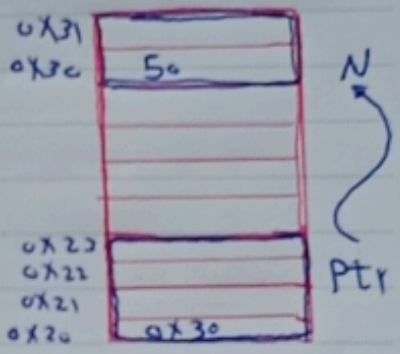
* Pointer :- ⇒ variables that hold addresses #

↳ Syntax ⇒ Data_Types *Pointer_Name = addres

Note ⇒ Not Data of Pointer
It's Data of thing that pointer Point to it #
(Data Type)

int N = 50;
int *Ptr = &N;
must be same Data Type — address operator

⇒ assume Int ⇒ 2 byte



* Size of Pointer :-

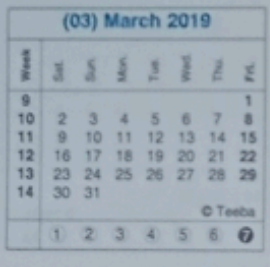
↳ Size of pointer is Constant and equal to address Bus depend on machine

int *Ptr1 = &N1;
Char *Ptr2 = &N2; ⇒ Size of Ptr1 or Ptr2 may be 4 byte depend on address Bus

* Reference (Vs) Dereference

1) Reference ⇒ Ptr = &N;
↳ put address of variable To Pointer #

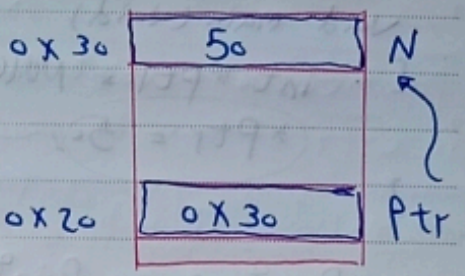
2) Dereference ⇒ *Ptr = 80;
↳ go to variable and Change it or Read it by Pointer



04 Monday March
27 Gamad El Akher 1440 Hijri

Note ⇒ int N = 50;
int *ptr;
ptr = &N;
≡ int N = 50;
int *ptr = &N; #

```
8
    Char N = 50;
9   Char *ptr = &N;
```




```
10  printf("%d", N ); ⇒ 50
    printf("%p", &N ); ⇒ 0x30
11  printf("%d", *ptr ); ⇒ 50
    printf("%p", ptr ); ⇒ 0x30
12  printf("%d", sizeof(ptr)); ⇒ 4 byte #
-   printf("%p", &ptr); ⇒ 0x20
1
```

* Wild pointer :-
2 ↳ Pointer has not been initialized to any thing #

```
3
    Void main(void)
4   { int N = 15;
      int *ptr;
5     *ptr = 700;
    }
```

wild pointer ⇒ local #
⇓
Initial value is Garbage
may be
- value Negative
- address in Range
- address not in Range

↳ * Solution for wild pointer :

1) init it by Null
*ptr = Null;

2) init it with address
int N = 5;
int *ptr = &N;

3) make pointer Global
init by default (zero) Null

* Null pointer:-

↳ pointer that point To Nothing or (NULL = 0x00)

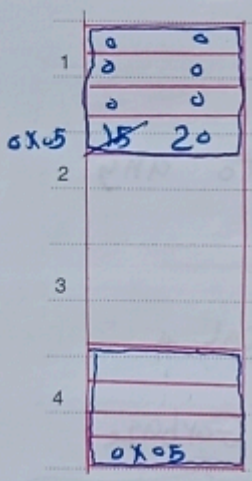
```
Void main (Void)
{   int *ptr = NULL;
    *ptr = 50;
}
```

→ run time error #

* operation on pointer:-

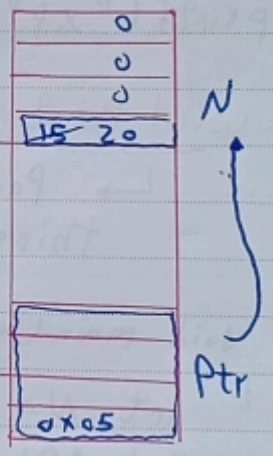
Data_Type *Ptr_Name;

1) Visibility 2) Step




```
Char N = 15;
Char *ptr = &N;
*ptr = 20;
```

```
int N = 15;
int *ptr = &N;
*ptr = 20;
```



#

1) adding or subtracting an integer from pointer
2) Comparing Pointer
3) Subtracting Pointer from Pointer
4) adding Pointer to Pointer X

```
Char N1 = 5 ; Char N2 = 10;
Char *ptr1 = &N1;
Char *ptr2 = &N2;
Char X = ptr1 + ptr2
```

error Not allowed #

(03) March 2019

Week	Sat.	Sun.	Mon.	Tue.	Wed.	Thu.	Fri.
9							1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29
14	30	31					

© Teeba

1 2 3 4 5 6 7

1) adding or subtracting an integer from pointer

```
int arr[ ] = { 1, 2, 3, 4 };
int *ptr = arr;
printf("%d", *ptr);   ⇒ (1) #
ptr = ptr + 1;
printf("%d", *ptr);   ⇒ (2) #
ptr++;
printf("%d", *ptr);   ⇒ (3) #
ptr--;
printf("%d", *ptr);   ⇒ (2) #
ptr = ptr - 1;
printf("%d", *ptr);   ⇒ (1) #
```

2) Comparing Pointer

```
int arr[ ] = { 1, 2, 3, 4 };
int *ptr0 = arr, *ptr1 = arr+1, *ptr2 = arr+2;
printf("%d", ptr2 > ptr0);   ⇒ (1) #
printf("%d", ptr2 > ptr1);   ⇒ (1) #
printf("%d", ptr2 < ptr1);   ⇒ (0) #
```

3) Subtracting pointer from pointer

```
char N1 = 17;
char N2 = 20;
char *ptr1 = &N1;
char *ptr2 = &N2;
char X = ptr1 - ptr2;
printf("%d", X);   ⇒ (1) #
```

↳ Step #

30 Gamad El Akher 1440 Hijri

* Pointer with Functions:

Pass by Value

Parameter of function is Copy of variable

```
#include "stdio.h"
void swap(int x, int y);
void main(void)
{ int a=10, b=20;
  swap(a, b);
}
void swap(int x, int y);
{ int temp=0;
  temp=x; x=y;
  y=temp;
}
```

Pass by address

Parameter of function is pointer point To variable address #

```
#include "stdio.h"
void swap(int *x, int *y);
void main(void)
{ int a=10, b=20;
  swap(&a, &b);
}
void swap(int *x, int *y)
{ int temp=0;
  temp=*x; *x=*y;
  *y=temp;
}
```

#

Note ⇒ Function Normally return only one value #

↳ by pointer it can return more than one value #

without pointer

```
int sum(int x, int y)
{ int sum=0;
  sum=x+y;
  return sum;
}
```

#

with pointer

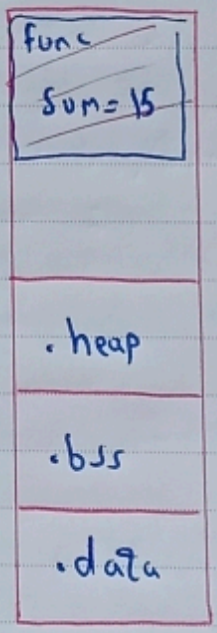
```
void calc(int x, int y, int *sum
          int *sub)
{
  *sum=x+y;
  *sub=x-y;
}
```

03 Ragab 1440 Hijri

* Function ⇒ Take Parameter and return address

↳ Syntax ⇒ Data_Type *ptr_Name(Parameter);

```
int *func(int x, int y);
void main(void)
{ int a=5, b=10;
  int *ptr = func(a, b);
  printf("%d", *ptr);
}
int *func(int x, int y)
{ static int sum = 0;
  sum = x + y;
  return &sum;
}
```



← Stack

⇒ it can be deleted from memory so we used static keyword

* Dangling Pointer :⇒ Pointer that point to memory location that has been deleted #

↳ Cases

1) Function Call

```
int *func(int x, int y)
{
  int sum = 0;
  sum = x + y;
  return &sum;
}
```

2) out of Scope

```
void main(void)
{
  int *ptr;
  { int x = 50;
    ptr = &x;
  }
}
```

#

04 Ragab 1440 Hijri

* Pointer To function :- ⇒ Pointer that point To a function #

↳ Syntax ⇒ Return Data-Type (*Ptr_Name)(Func_parameter);

initilization ⇒ Ptr_Name = Func_Name;

Calling ⇒ Ptr_Name (Func Parameter); #

```
Void Sum(int x, int y, int *Sum);
Void (*Fptr)( int x, int y, int *Sum);

Fptr = Sum;
```

⇒ Name of function is a Constant Pointer that point to first addres of func #

```
Void main(void)
{
    int Sum = 0;
    Fptr( 5, 2, & Sum);
}
```

#

⇒ Call Back function #

* Pointer with array :

⇒ Passing Array to function #

```
Void func( int arr[], int size);
Void main ( void )
{  int arr[ ] = { 1, 2, 3, 4};
    func( arr, 4);
    printf( "%d", arr[2]);
}
Void func( int arr[], int Size)
 {
    arr[2] = 7;
 }
```

#

Name of array is Constant Pointer first addres point to in array #

05 Ragab 1440 Hijri

* Pointer To Array ⇒ Pointer that point To first element of array #

↳ Syntax ⇒ DataType *Ptr_Name = arr_Name; #

```
int arr_Sum( int *ptr, int size);
Void main (void)
{ int arr[] = { 1, 2, 3, 4, 5}, Sum = 0;
  Sum = arr_Sum( arr, 5);
  Printf ("%d", Sum);
}
int arr_Sum ( int *ptr, int size)
{  for (int i=0; i < size; i++)
   {  Sum += ptr[i];
   }
} return Sum;
```

Note ⇒ int *ptr = arr; (address #)

arr = &arr = &arr[0] = arr+0 = ptr #

&arr[1] = arr+1 = ptr+1

Value # ⇐ *(&arr[1]) = *(arr+1) = *(ptr+1)

arr++ ⇒ Not allawed

Constant Pointer #

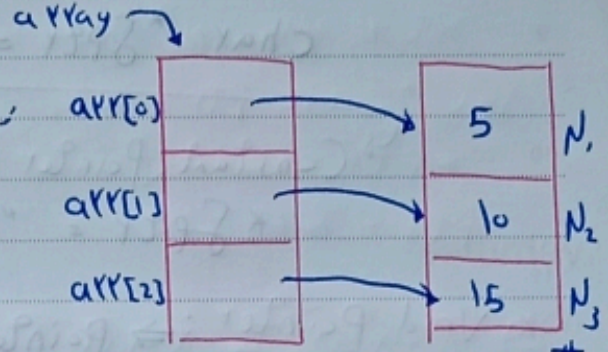
(03) March 2019

Week	Sat.	Sun.	Mon.	Tue.	Wed.	Thu.	Fri.
9							1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22

* Array of Pointers :- ⇒ an array that all element is pointers #

↳ Syntax ⇒ Data_Types *array_Name [element] ;

```
int * arr [3] ;
int N1 = 5, N2 = 10, N3 = 15;
    arr [0] = & N1 ;
    arr [1] = & N2 ;
    arr [2] = & N3 ;
```



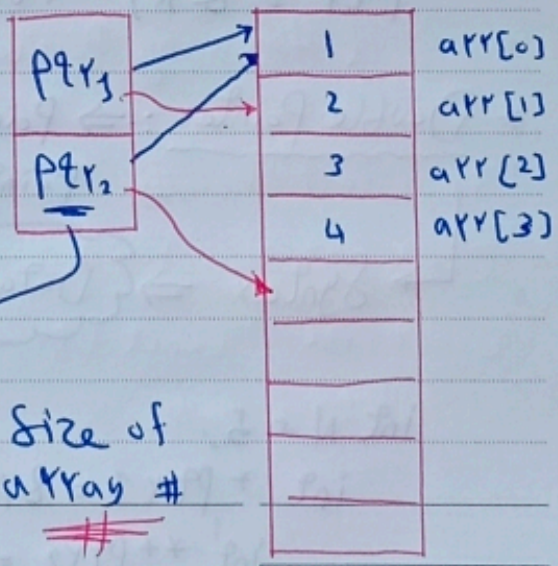
#

* Pointer To array of n element :

↳ Pointer that Point To all element of Array #

↳ Syntax ⇒ Data_Types (*Ptr_Name) [element] ;

```
Void Main (Void)
{
  int *ptr1 ;
  int (*ptr2) [4] ;
  int arr [4] = {1, 2, 3, 4} ;
  ptr1 = arr ;
  ptr2 = arr ;
  ptr1++ ;
  ptr2 ++ ;
}
```



Step = Size of array #

March 14

07 Ragab 1440 Hijri

* Pointer To string ⇒ Pointer that point to string # (first character)

↳ Syntax ⇒ Data_Type *ptr_Name = "string";

char *sptr = "WelCome";

↳ Constant Pointer Can not edit #

*sptr = "Hello"; ⇒ run time error #

* Void Pointer :⇒ pointer thas has no access size or pte-Data information is known #

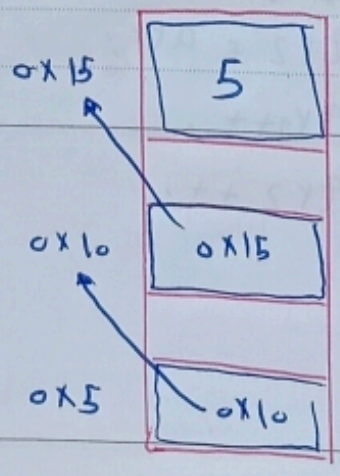
↳ Syntax ⇒ void *ptr_Name;

void *ptr;
int x = 5;
ptr = &x, Ptr = (int*)ptr; (Casting #)

* Double Pointer :⇒ Pointer that point To Pointer that Point To a variable #

↳ Syntax ⇒ Data_Type **ptr_Name; #

int N = 5;
int *ptr1 = &N;
int **ptr2 = &ptr1;



#

(03) March 2019

Week	Sat.	Sun.	Mon.	Tue.	Wed.	Thu.	Fri.
9							1
10	2	3	4	5	6	7	8
11	9	10	11	12	13	14	15
12	16	17	18	19	20	21	22
13	23	24	25	26	27	28	29
14	30	31					

© Teeba

10 Ragab 1440 Hijri

* Pointer with Constant:-

1) Const Pointer To Data_Type

```
int N1 = 10, N2 = 50;
int *Const ptr = &N1;
*ptr = 20;   ✓
ptr = &N2;   ✗
```

Can change value of Variable that Pointer Point To it But Can not Change addres #

2) Pointer To Const Data_Type

```
Const int N1 = 10, N2 = 50;
int *ptr1 = &N1;
*ptr1 = 20;   ✗
ptr1 = &N2;   ✓
```

Pointer Can be Change addres that it point to it

⇒ May be change value of Constant Variable or may be Not #

(Hacking Pointer)

3) Const Pointer To Const Data_Type

```
Const int N1 = 10, N2 = 50;
Const int *Const ptr = &N1;
ptr = &N2;   ✗
*ptr = 30;   ✗
```

Can not change addres or value that pointer point to it #

4) Pointer To Const Data_Type

```
Const int N1 = 10, N2 = 50;
Const int *ptr = &N1;
   *ptr = 50;   X
   ptr = &N2;   ✓
```

can change address that point to it, but can not change value #

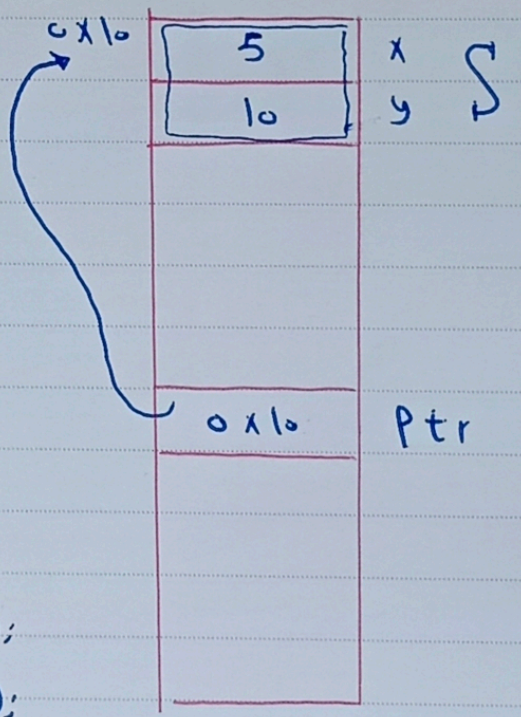
* Pointer with struct :-

```
Typedef struct
{  int x;
   int y;
}S;

void main(void)
{  S A = {5, 10};
   S *ptr = &A;
   printf("%d", Ptr -> x);
   printf("%d", ptr -> y);
}
```

arrow operator

#



#