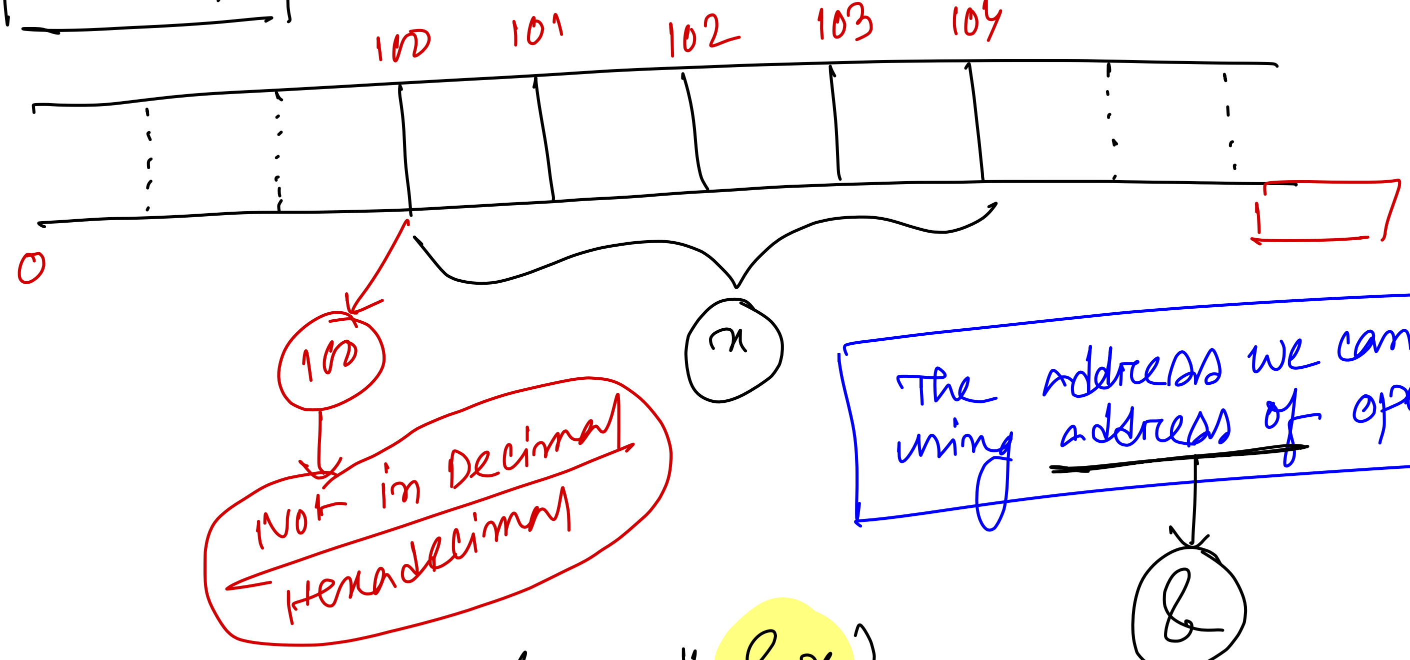
Pointers

1. Simulating call by reference.
2. Returning more than one value from a function.
3. Accessing Dynamically Allocated memory.
4. Implementing Data Structures like linked list, stack, queue, tree, graph etc.
5. Improving Efficiency.

```
int x;
```



The address we can find using address of operator → $\&$

scanf("%d", &x) → The address of x

%p is used to print the address. → Every time it may differ as it depends on the OS

%p → format specifier

$\&j \longrightarrow$ valid

$\boxed{\&289} \longrightarrow$ Invalid

$\&(\underline{j+k}) \longrightarrow$ Invalid.

↓

Constant

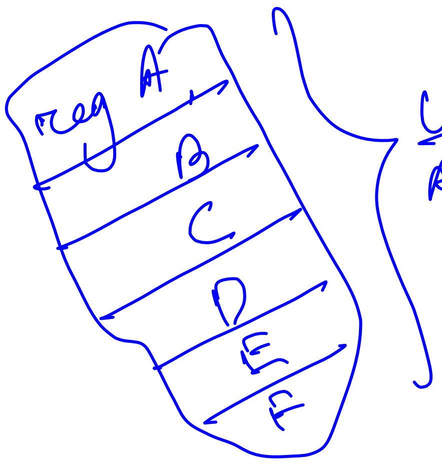
j=20, k=30

(j+k) &(50)

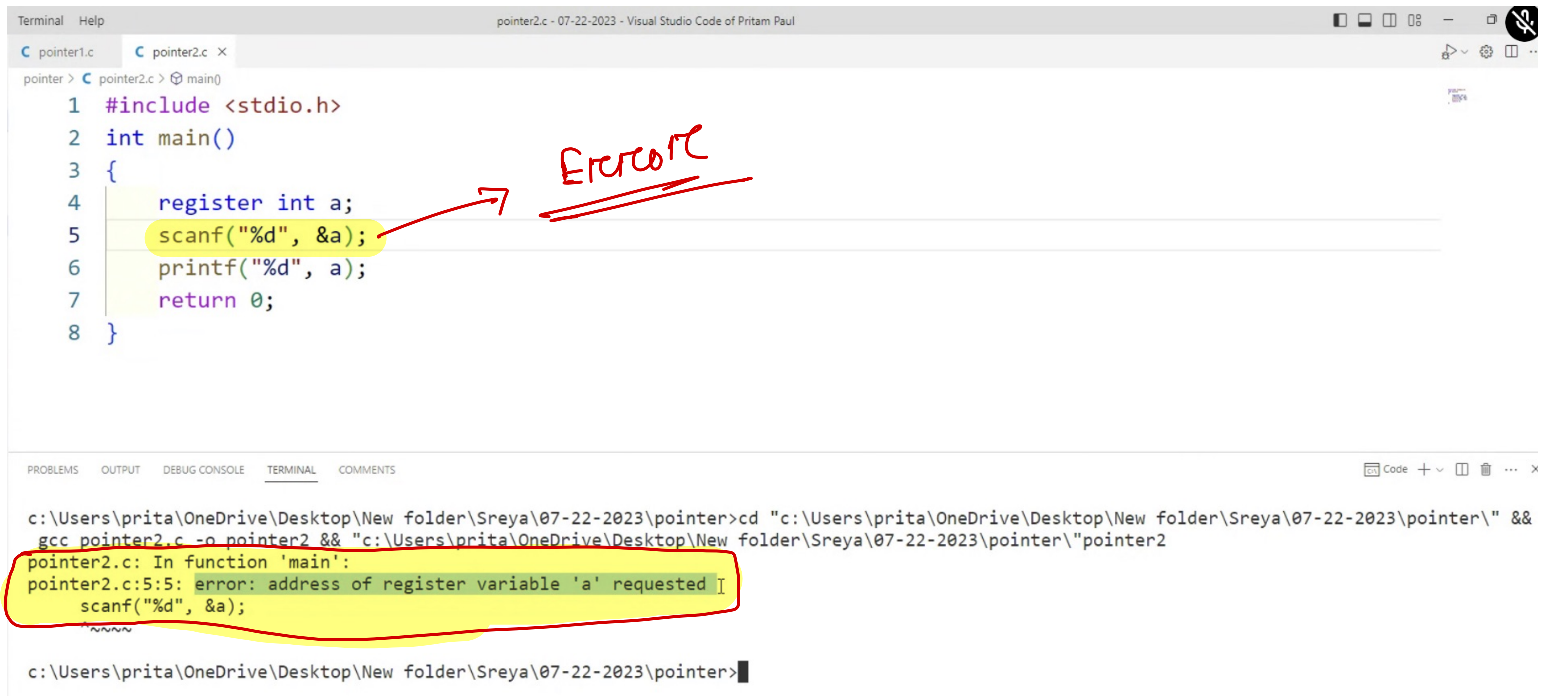
$\&$ r_var; $\longrightarrow$ r_var in register variable

↓

Invalid



Constant Address.


Terminal Help
pointer2.c - 07-22-2023 - Visual Studio Code of Pritam Paul
pointer1.c
pointer2.c
pointer > pointer2.c > main()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL COMMENTS
Code

```c
#include <stdio.h>
int main()
{
    register int a;
    scanf("%d", &a);
    printf("%d", a);
    return 0;
}
```

Error

```
c:\Users\prita\OneDrive\Desktop\New folder\Sreya\07-22-2023\pointer>cd "c:\Users\prita\OneDrive\Desktop\New folder\Sreya\07-22-2023\pointer\" &&
 gcc pointer2.c -o pointer2 && "c:\Users\prita\OneDrive\Desktop\New folder\Sreya\07-22-2023\pointer\"pointer2
pointer2.c: In function 'main':
pointer2.c:5:5: error: address of register variable 'a' requested
     scanf("%d", &a);
     ^~~~~

c:\Users\prita\OneDrive\Desktop\New folder\Sreya\07-22-2023\pointer>
```

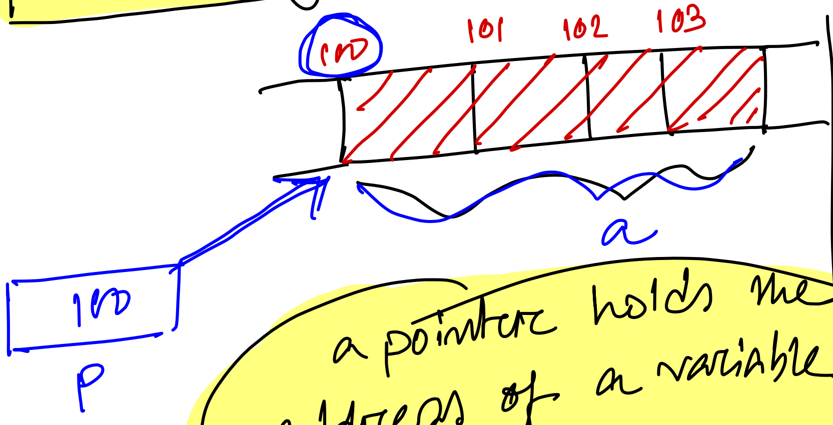
$\&$ $\longrightarrow$ Also called as reference operator.

↓

$\boxed{\text{we can use the referent to point to that variable}}$

Pointer Variable

<return_type> * <variable_Name>;



a pointer holds the address of a variable

By which it can point to that variable

int * p;

p is a pointer which is pointing to an integer.

float * x;

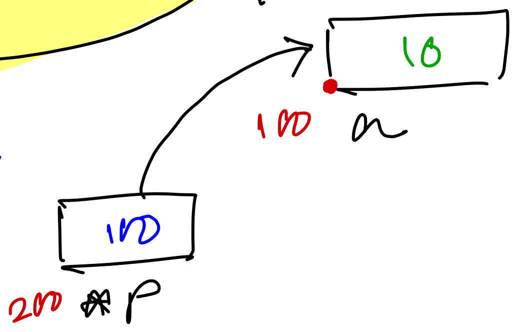
char * c;

int a = 10;

int * p;

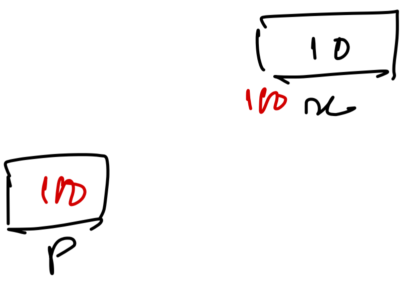
p = & a;

int * p = & a;



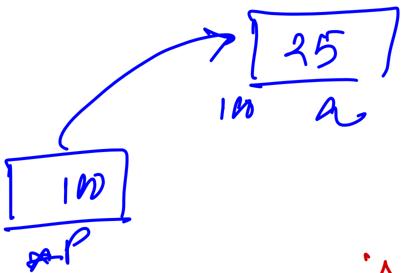
int x = 10;

int p = & x;



int a = 25;

int * p = & a;



int * p;

the type of p is (int *)

```
float f = 20.7;
float *p = &f;
```

```
char c = 'A';
char *cp = &c;
```

where as

$\text{sizeof}(f) \rightarrow 4$

$\text{sizeof}(c) \rightarrow 1$

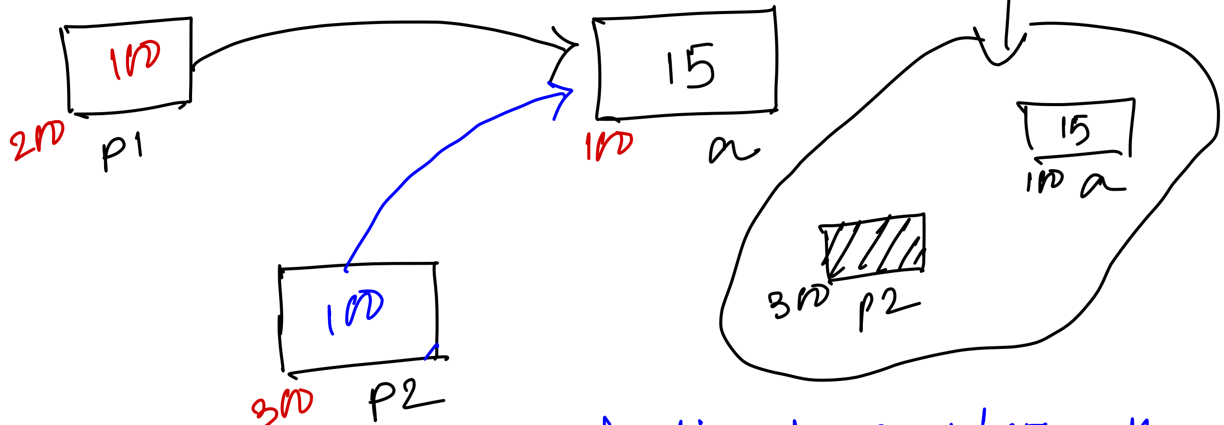
$\text{sizeof}(p) \longrightarrow 4$

$\text{sizeof}(cp) \longrightarrow 4$

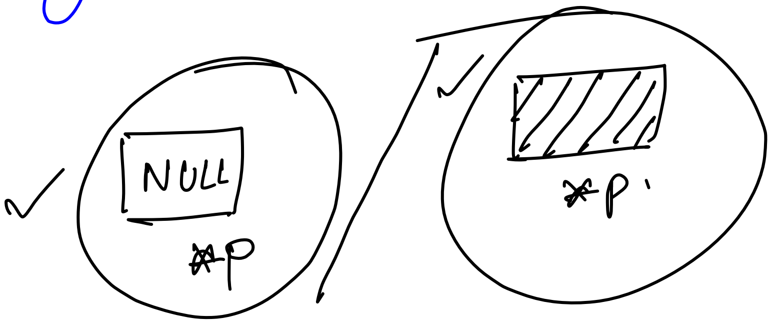
(*) More than 1 pointers can point to same variable.

```
int a = 15;
int *p1 = &a;
int *p2 = &a;
```

p2 = NULL;

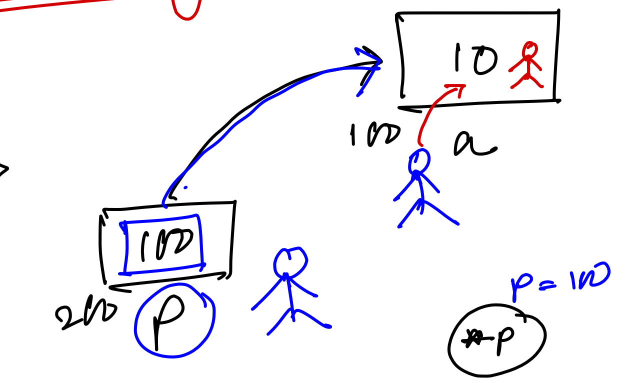


(*) when a pointer is not pointing to anywhere then we can say that it is pointing to NULL.



int *p; → dereferencing operatore.

```
int a=10;
int *p=&a;
```



```
printf("%d", a);
```
10

```
printf("%d", *p);
```
10

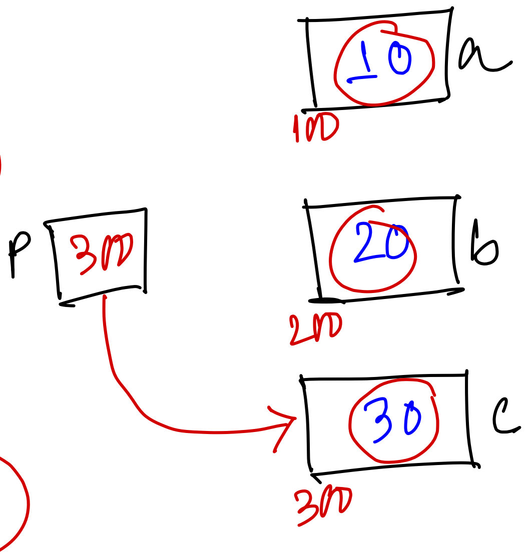
```
printf("%p", p);
```
100

$*p \rightarrow \cancel{*}(\cancel{\&}a)$

```
int a=10, b=20, c=30;
int *p=&a;
printf("%d", *p);
```
→ 10

```
p=&b;
printf("%d", *p);
```
→ 20

```
p=&c;
printf("%d", *p);
```
→ 30



```
int a=10;
float b=4.7;
int *p=&b;
float *q=&a;
```

Bcz

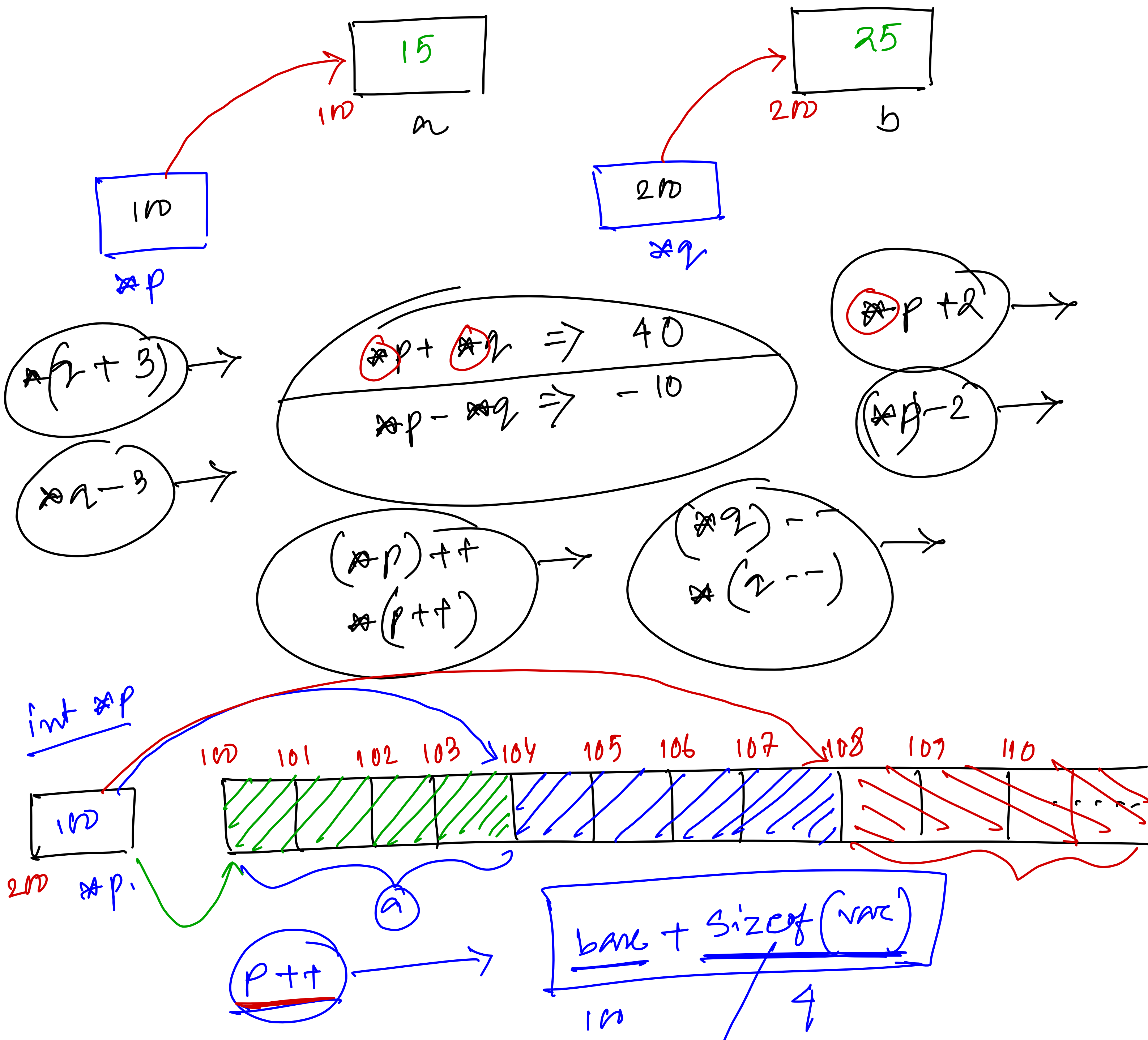
int *p → p in a pointere to integere

float *q → q in a pointere to float.

Pointer Arithmetic

All types of operation are not possible.

1. Addition of an integer to a pointer and increment operation.
2. Subtraction of an integer from a pointer and decrement operation.
3. Addition of 2 pointers.
4. Subtraction of 2 pointers.



$p + = 2$

$\boxed{\text{base} + (2 * 4)}$

(*) The arithmetic operation that can not be perform-ed on pointers are :-

(1) Addition, Multiplication, Division of two pointers
(2) Multiplication between pointer and any number.
(3) Division of a pointer by any number.
(4) Addition of float or double values to pointer

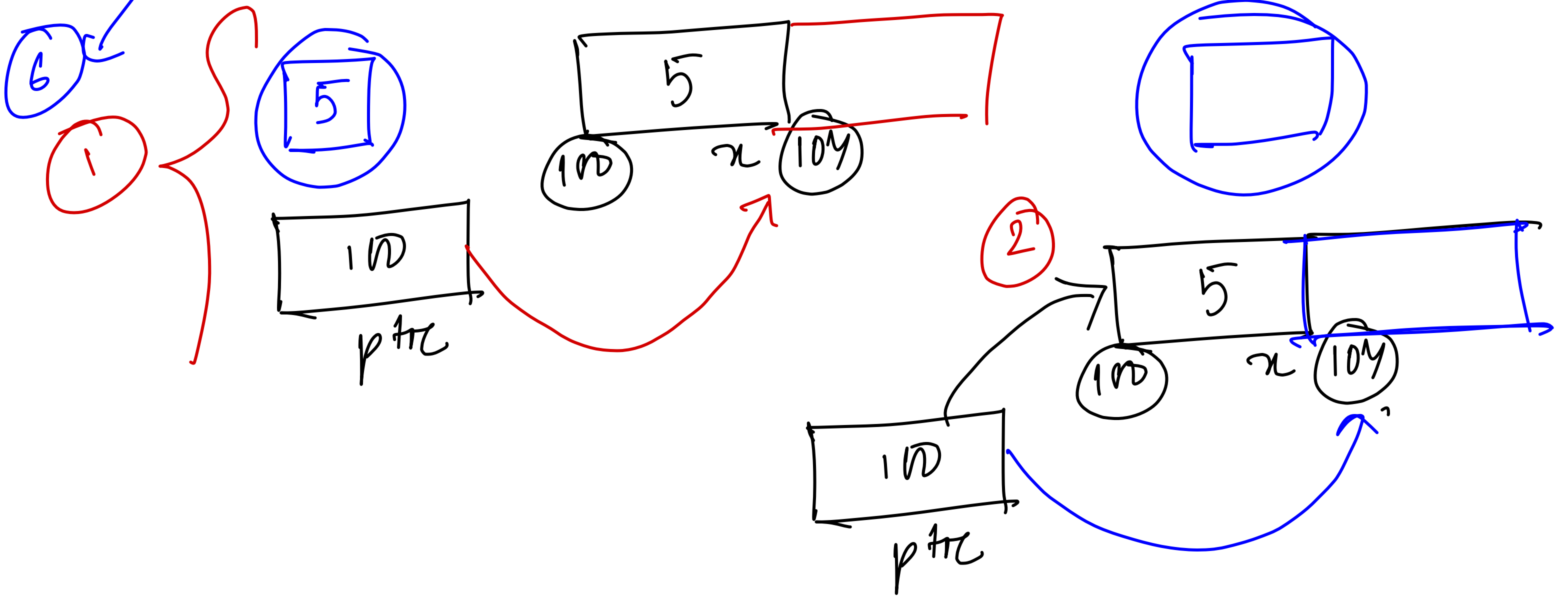
Precedence of Dereferencing operator and Inc./Dec. operators.

① $x = (*(ptr++))$;

② $x = (*(++ptr))$

③ $x = (++(*ptr))$

④ $x = (*ptr)++$



6
1
5
5
100
x
104
100
ptr
2
5
100
x
104
100
ptr