CS & IT ENGINEERING

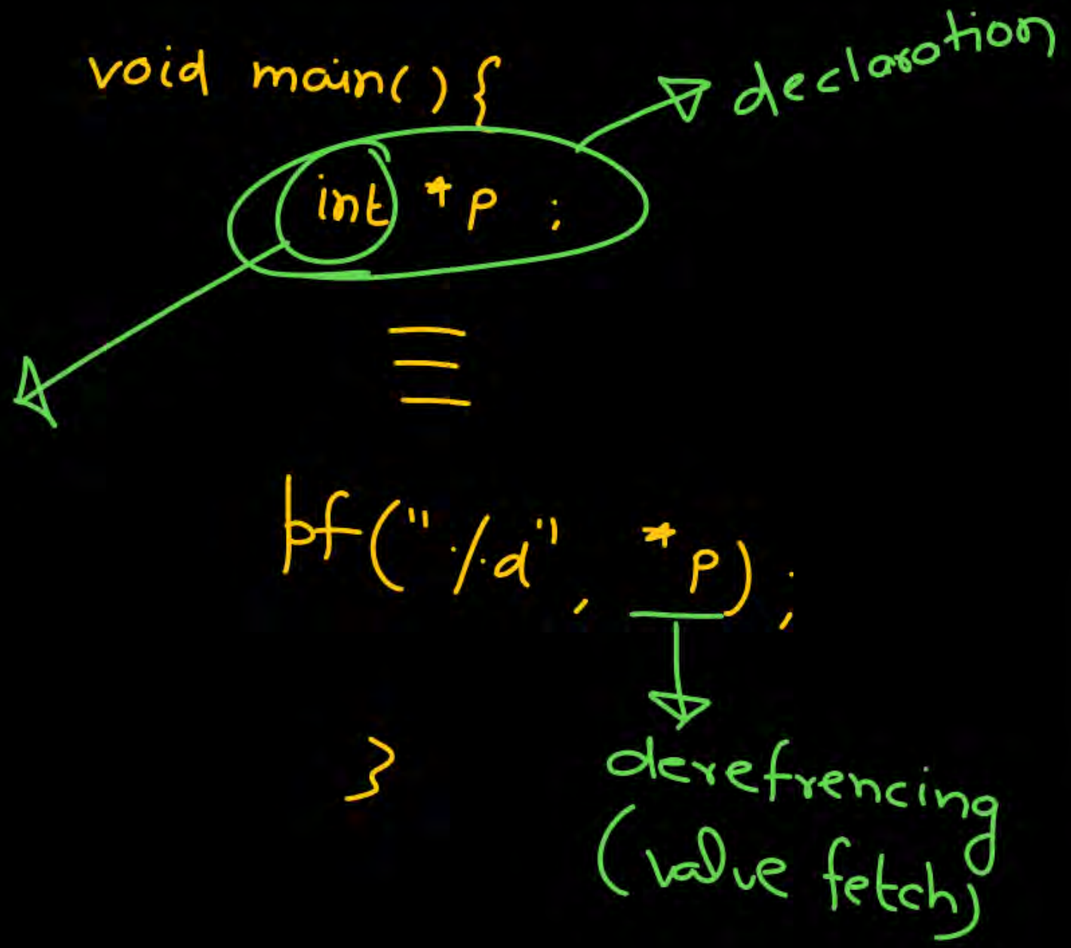
C Programming

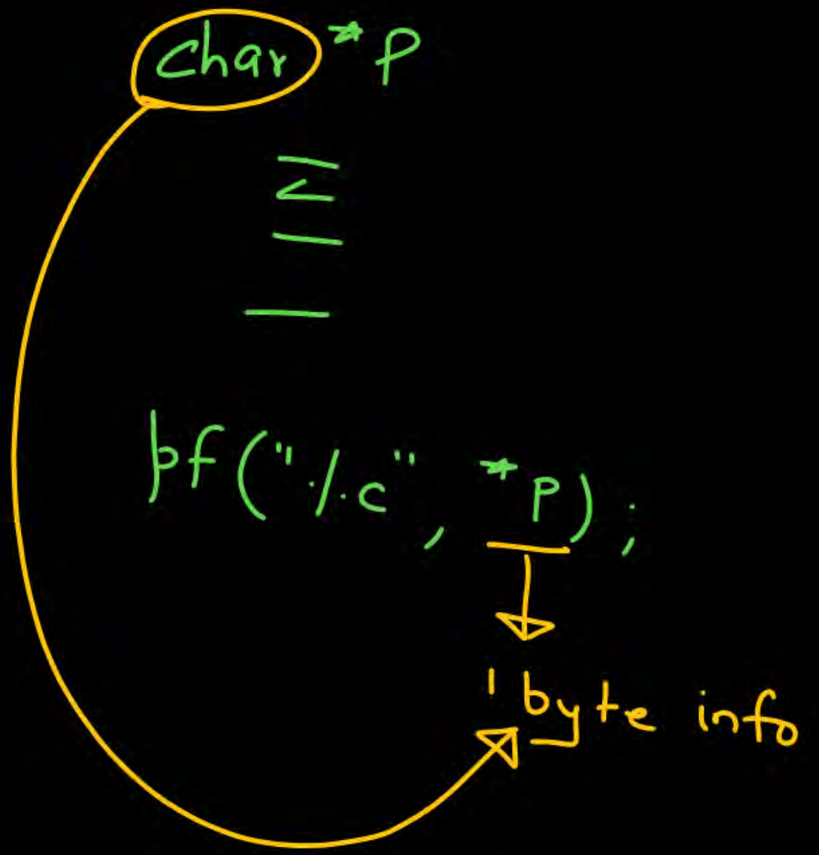
Arrays and Pointers

Lec - 08

By- Pankaj Sharma Sir

Arrays and Pointers (Part- 08)


void main( ) {
declaration
int *p ;
≡
bf(" %d", *p);
}
derefrencing
(value fetch)

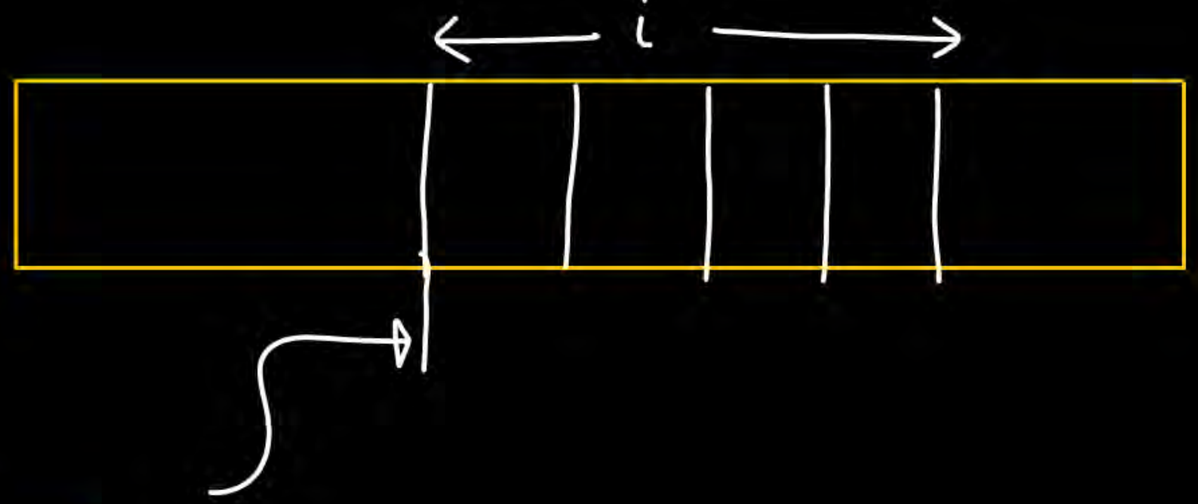

char *p
≡
bf(" %c", *p);
1 byte info

void pointer

`void *Ptr;` → Address

```
int  i  = 369;
char ch = 'A';
```

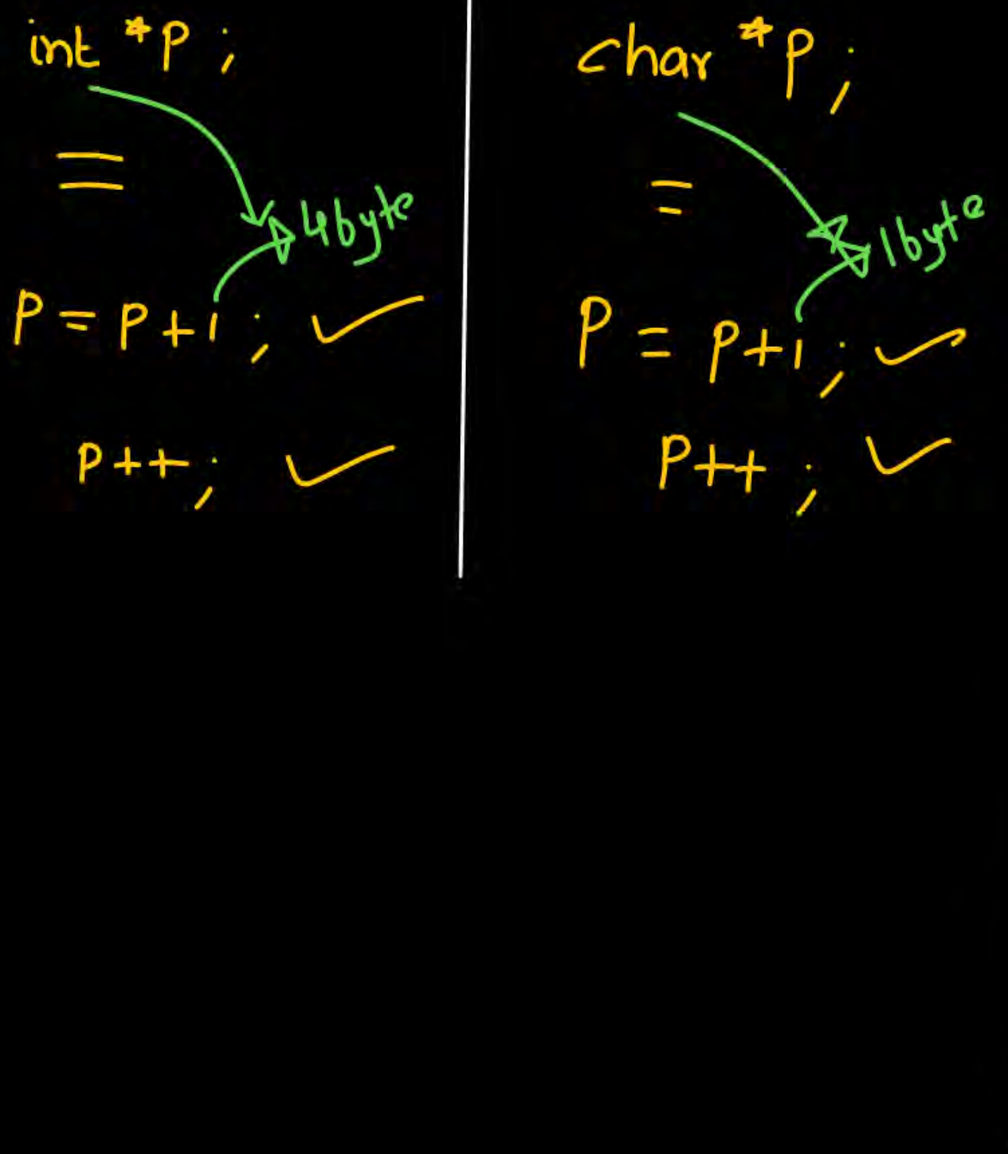
`Ptr = &i;` ✓



Error

bcz

```
pf("%d", *Ptr);
pf("%d", *(int*)Ptr);
```


int *p ;
4byte
P = P+1 ;
P++ ;
char *p ;
1byte
P = P+1 ;
P++ ;
void *p ;
P++ ;
P = P+1 ;
P = P-1 ;
Ud ke laal
marega

```
int a = 20;
char ch = 'A';
void *p;
p = &a;
pf("%d", *(int*)p);
p = &ch;
pf("%c", *(char*)p);
```

20

A

(i) Don't try to dereference any void pointer without typecasting

(ii) Do not apply arithmatic operations on void pointer.

NULL Pointer → implement

* Specially designed pointer

float

H.NO – 60.2583

Mathura

Valid pointer

Valid address → -ve x

Valid address → float x

H.NO : -87

Agra → -ve x

Valid address → unsigned int

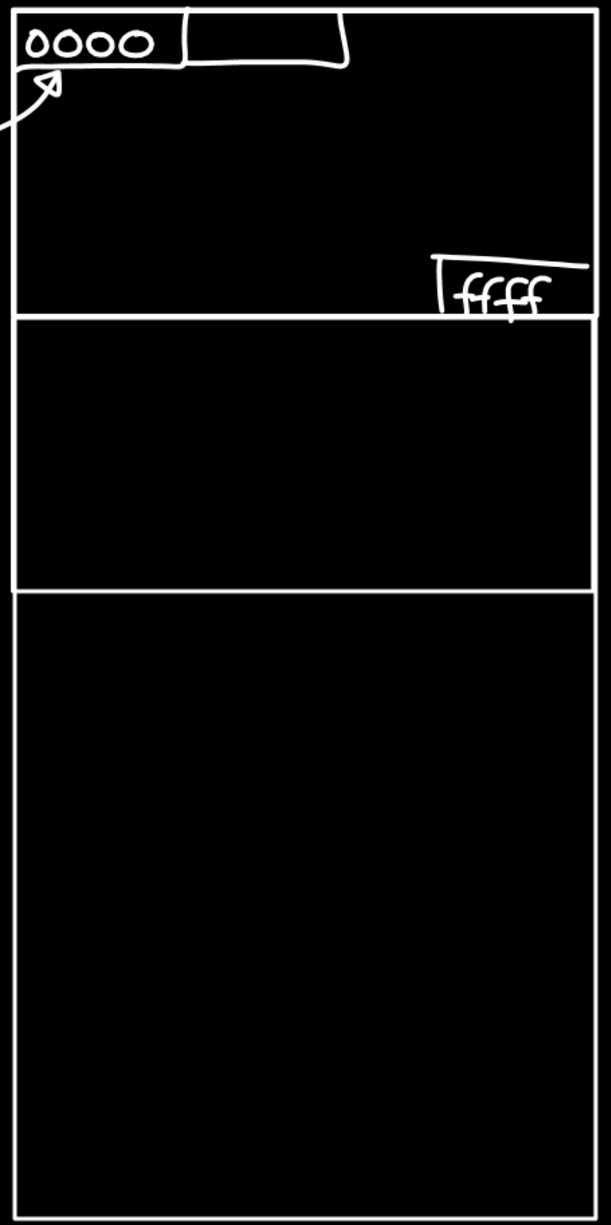
NULL pointer

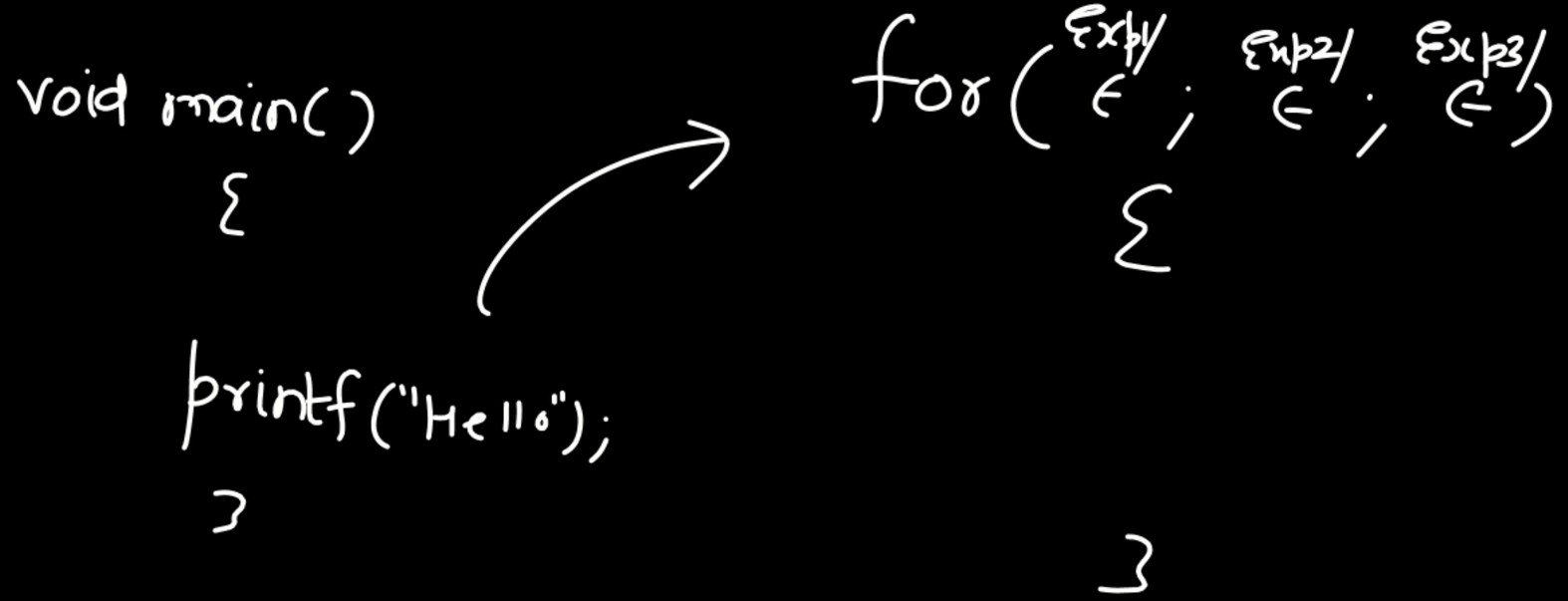
```
int *P = (int*)0 ;
```

(ii)

0 → false

```
if(NULL)        if(0)
  {               {


  }               }
```

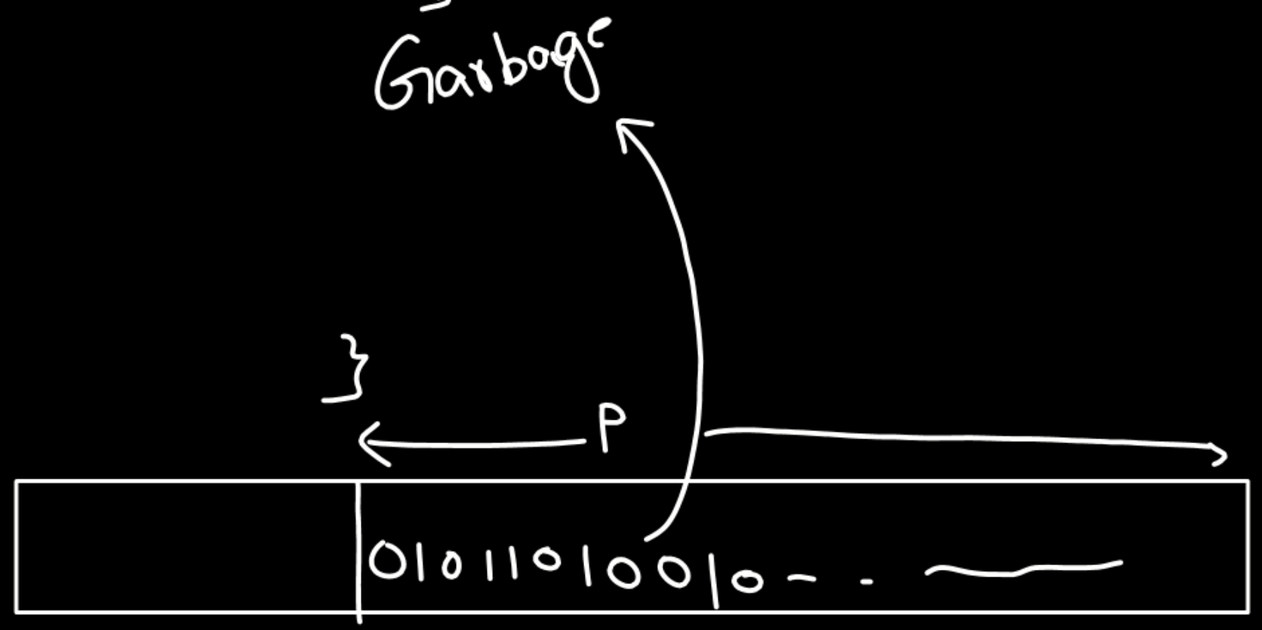



void main()
{
printf("Hello");
}
for(Exp1/ ∈ ; Exp2/ ∈ ; Exp3/ ∈)
{
}

Wild Pointer

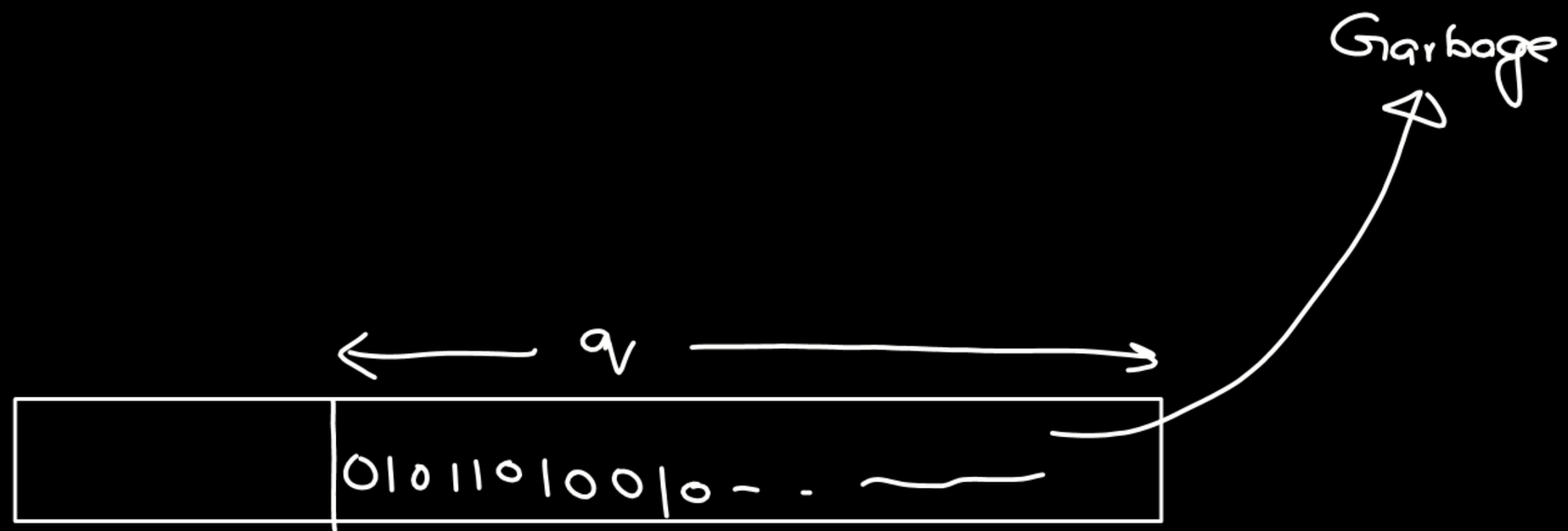
```
void main(){
    int p;
}
```

```
void main(){
    int *q;
}
```



Wild Pointer

```
void main(){
    int *qr;
```

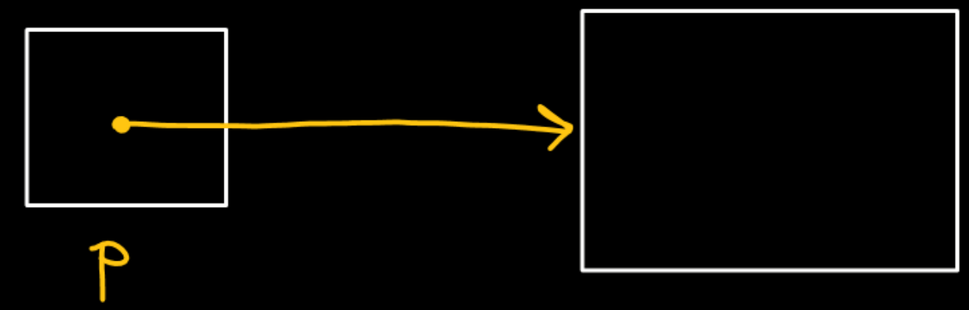


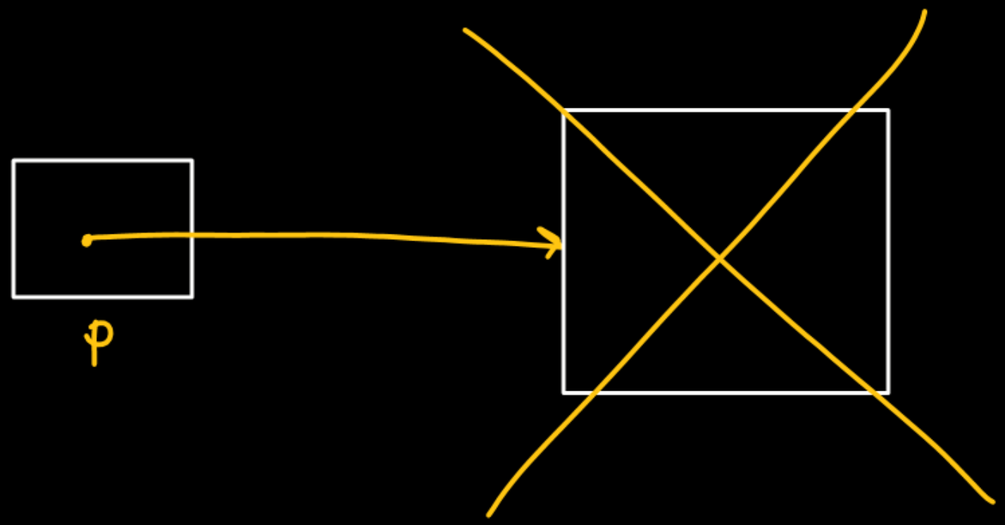
```
void main(){
    int x = 1008;
    int *p;

    *p = 10;
```

Process

Wild pointer

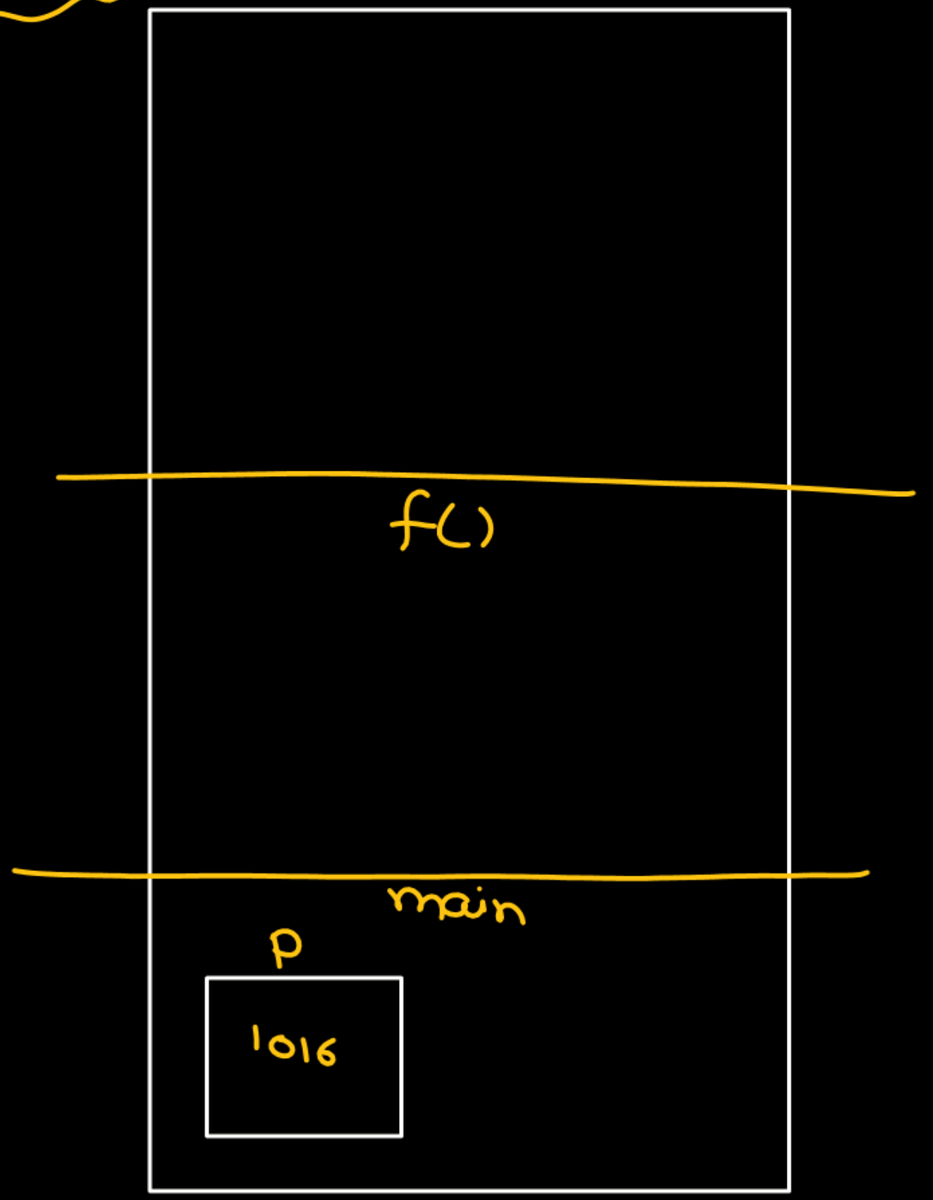
Dangling Pointer



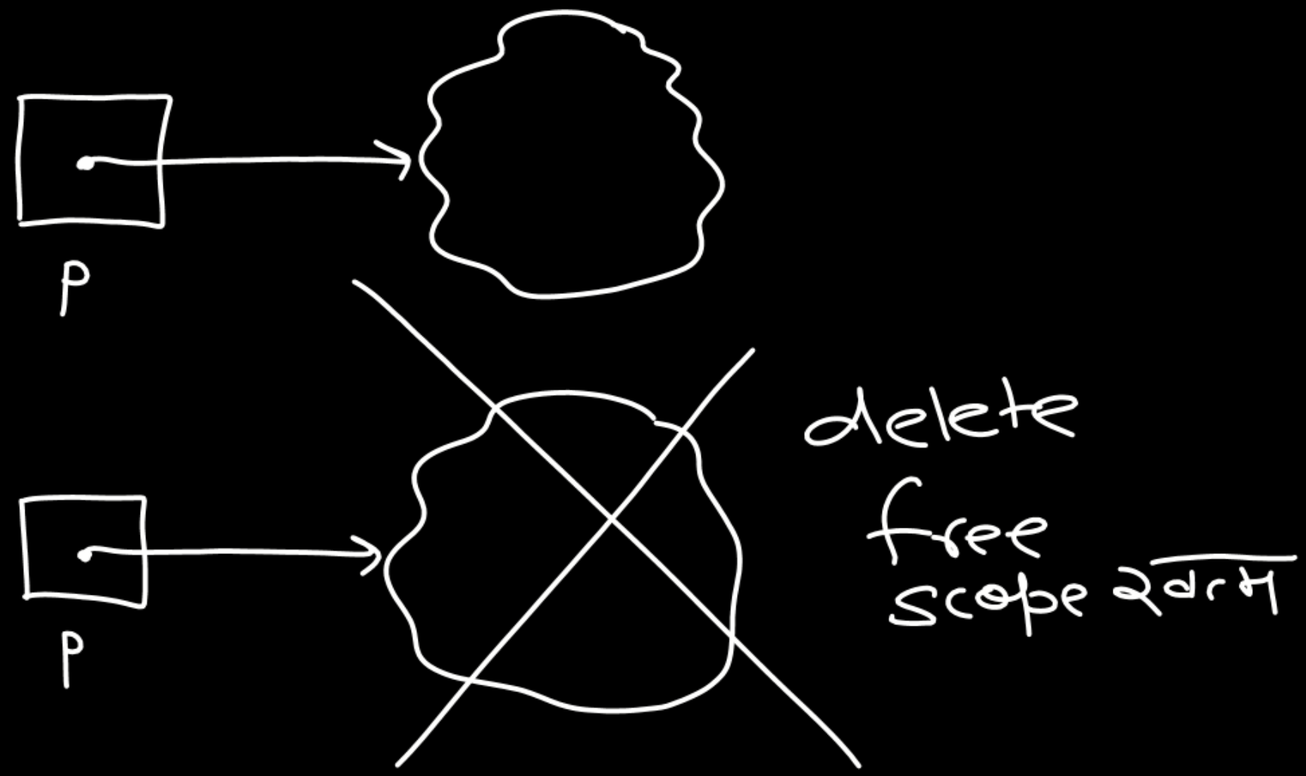


DS → clear

```
int *f(){
 static int a = 10;
        return &a;
          }

void main(){
          int *p;
          p = f();
          printf("%d", *p);
           }
```



1.) void pointer
2.) NULL pointer
3.) Wild pointer
4.) Dangling pointer

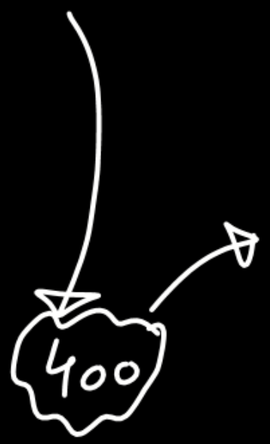


Dynamic Memory Allocation

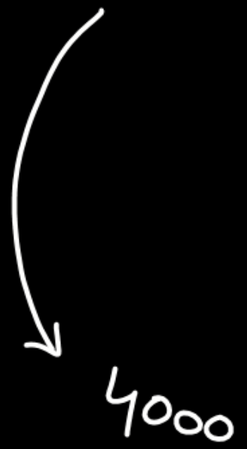
Heap

array

(i) `int a[4000];`



(ii) `int a[400];`



(i) malloc
(ii) calloc
(iii) realloc
(iv) free

] used for DMA in C

malloc

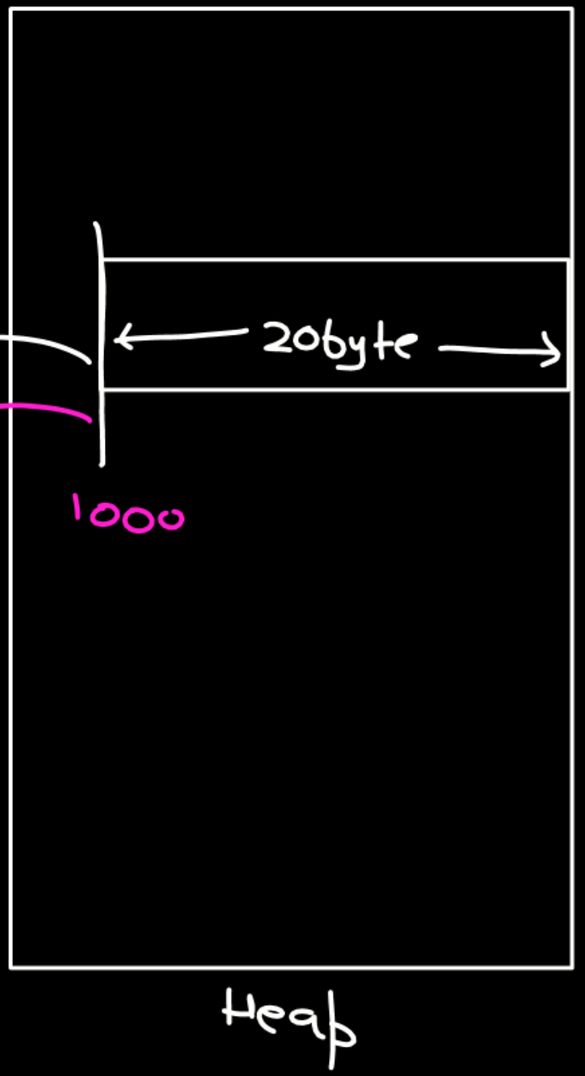
malloc(20)

size in bytes

(void *) malloc(size in byte)

Pointer

starting address return



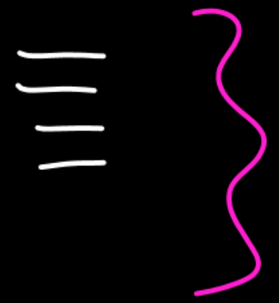
(void *) malloc (size_t size)

unsigned int

Syntax

stu int 2 byte s integer

```
int *p;
P = malloc(10);
```



stu int — 4 byte

दोनों में दायरा

```
int *p;
P = malloc(20);
```

Heap

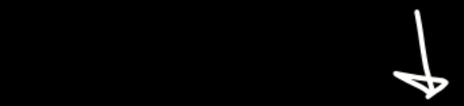
malloc(100000)

NULL

block

i) Search

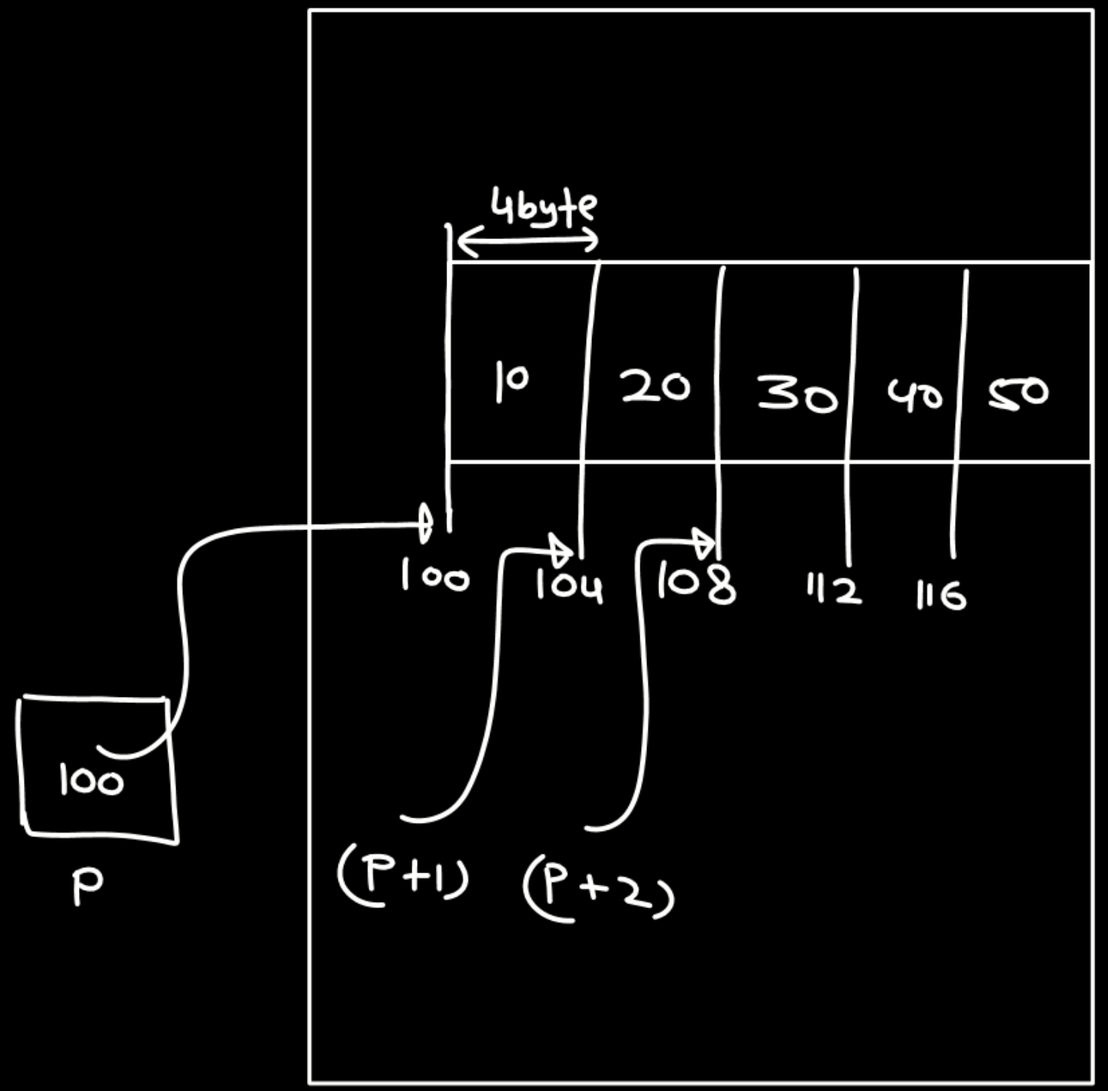
memory available



Sizeof

```
int *P;
P = malloc (5 x sizeof(int))
```

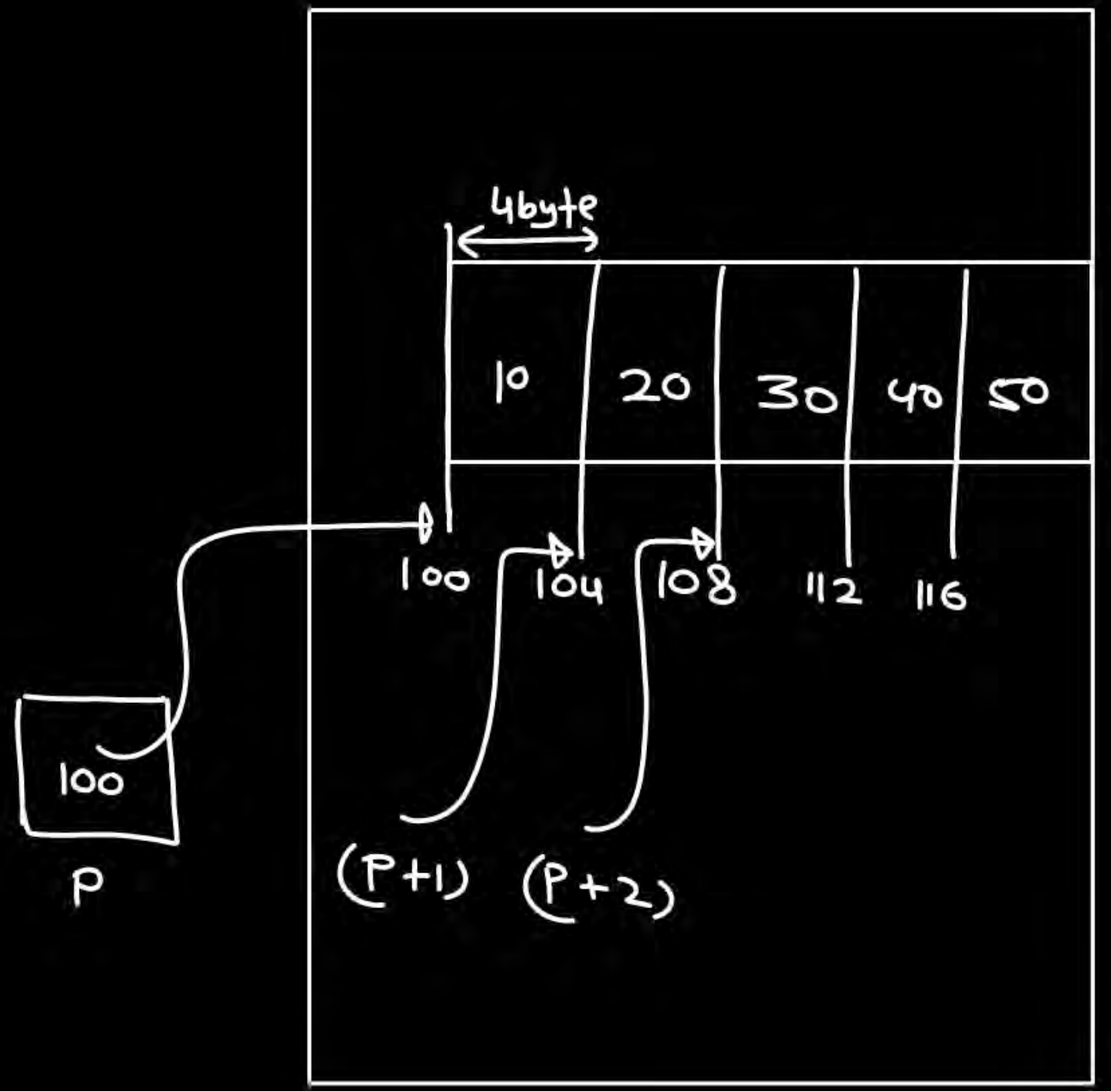
```
scanf ("%d", P);      10↵
scanf ("%d" P+1);     20↵
scanf ("%d", P+2);    30↵
scanf ("%d", P+3);    40↵
scanf ("%d", P+4);    50↵
```



Sizeof

```
int *P;
P = malloc (5 x sizeof(int))
```

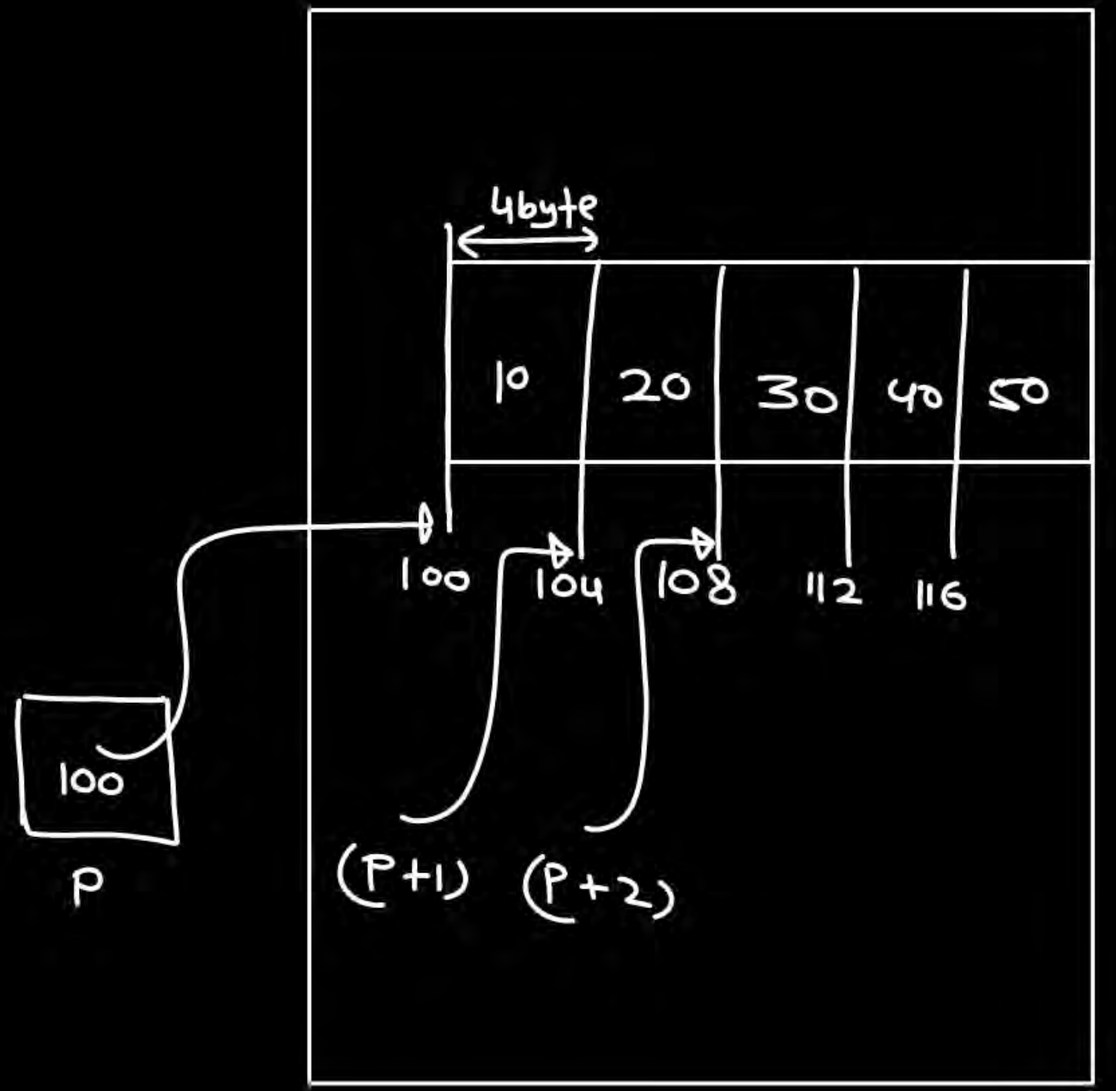
```
scanf("%d", P+0);     10↵
scanf("%d" P+1);      20↵
scanf("%d", P+2);     30↵
scanf("%d", P+3);     40↵
scanf("%d", P+4);     50↵
for(i=0; i<5; i+1)
  scanf("%d", P+i);
```



Sizeof

```
int *P;

P = malloc (5 x sizeof(int))

for(i=0; i<5; i++)
    scanf("%d", P+i);
```

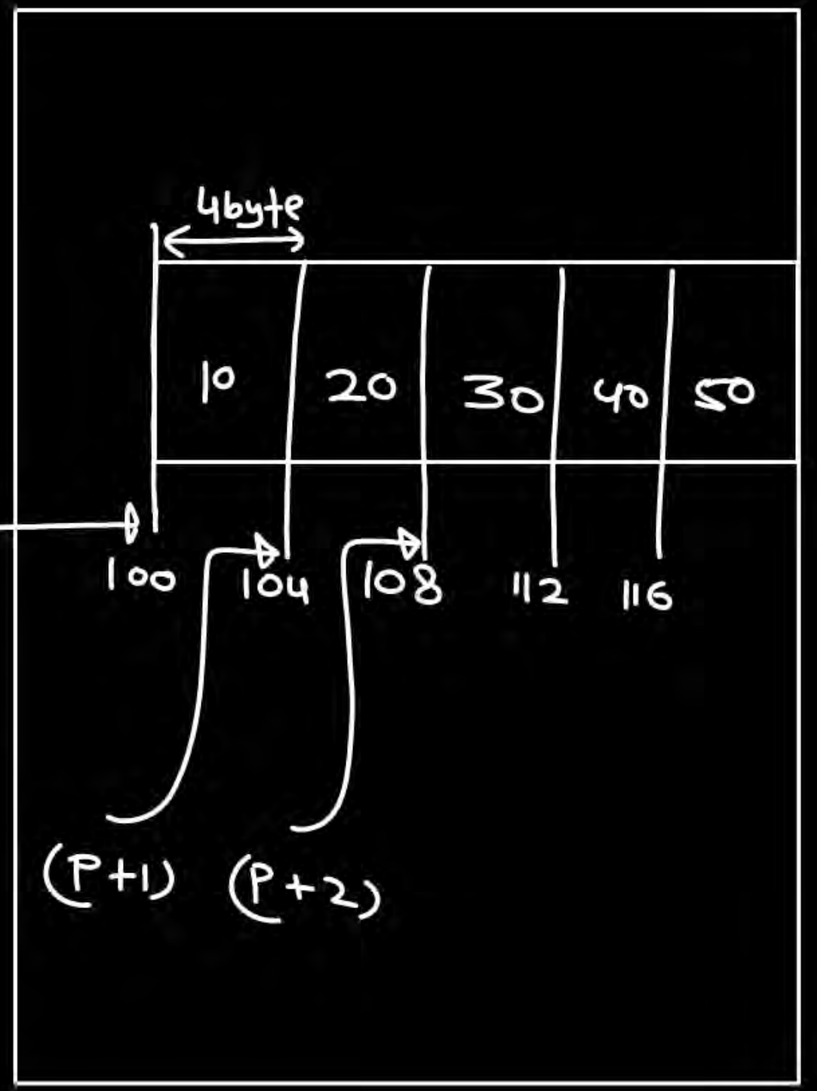
```
printf("%d", *P);      10
printf("%d", *(P+1));  20
printf("%d", *(P+2));  30
printf("%d", *(P+3));  40
printf("%d", *(P+4));  50
```



Sizeof

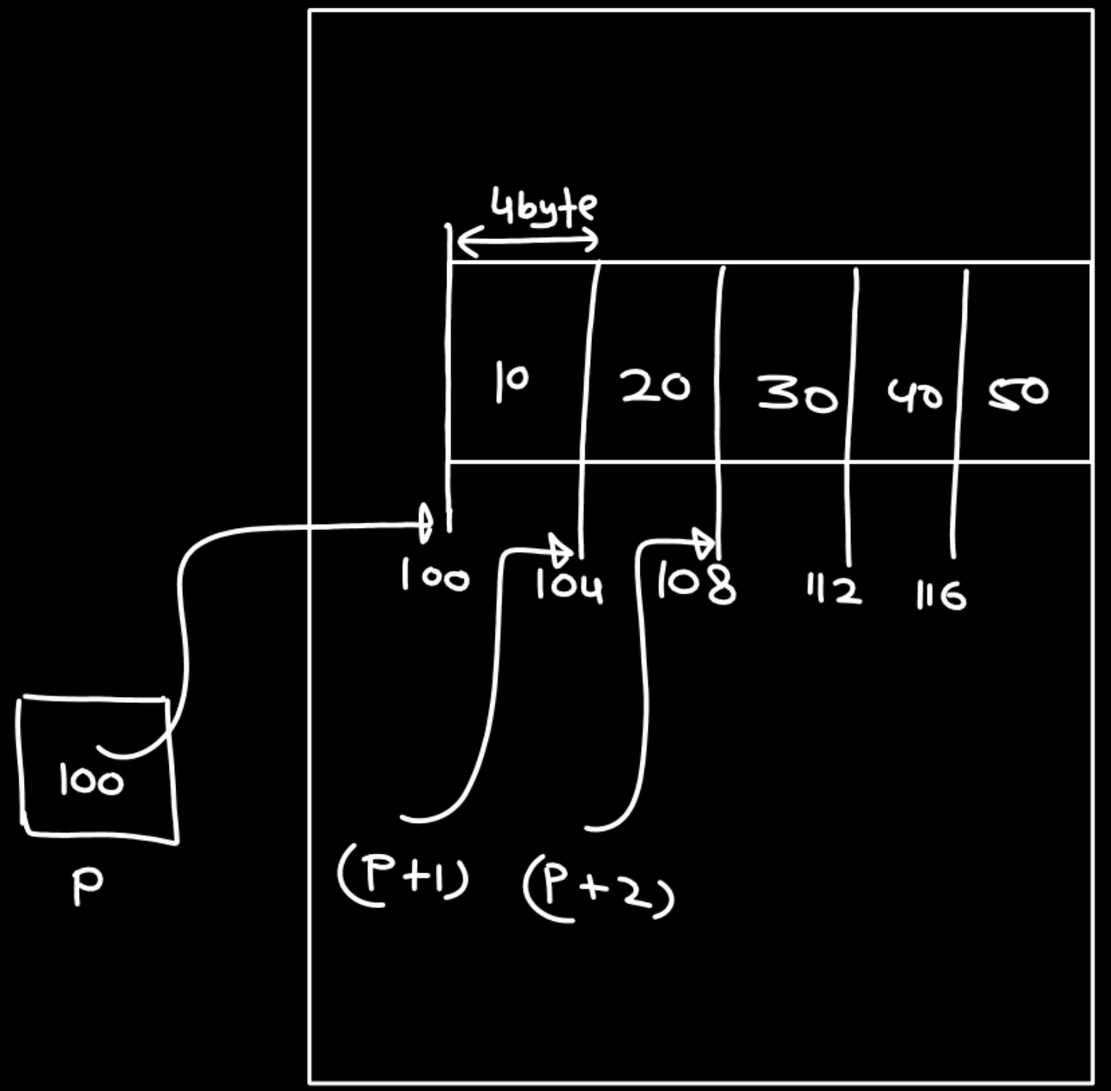
```
int *P;

P = malloc (5 x sizeof(int))

for(i=0; i<5; i++)
    scanf("%d", P+i);

for(i=0; i<5; i++)
    printf("%d", *(P+i));
```

```
printf("%d", *(P+0)); 10
printf("%d", *(P+1)); 20
printf("%d", *(P+2)); 30
printf("%d", *(P+3)); 40
printf("%d", *(P+4)); 50
```



Sizeof

```
int *P;

P = malloc (5 x sizeof(int))

for(i=0; i<5; i++)
    scanf("%d", P+i);

for(i=0; i<5; i++)
    printf("%d", *(P+i));
```

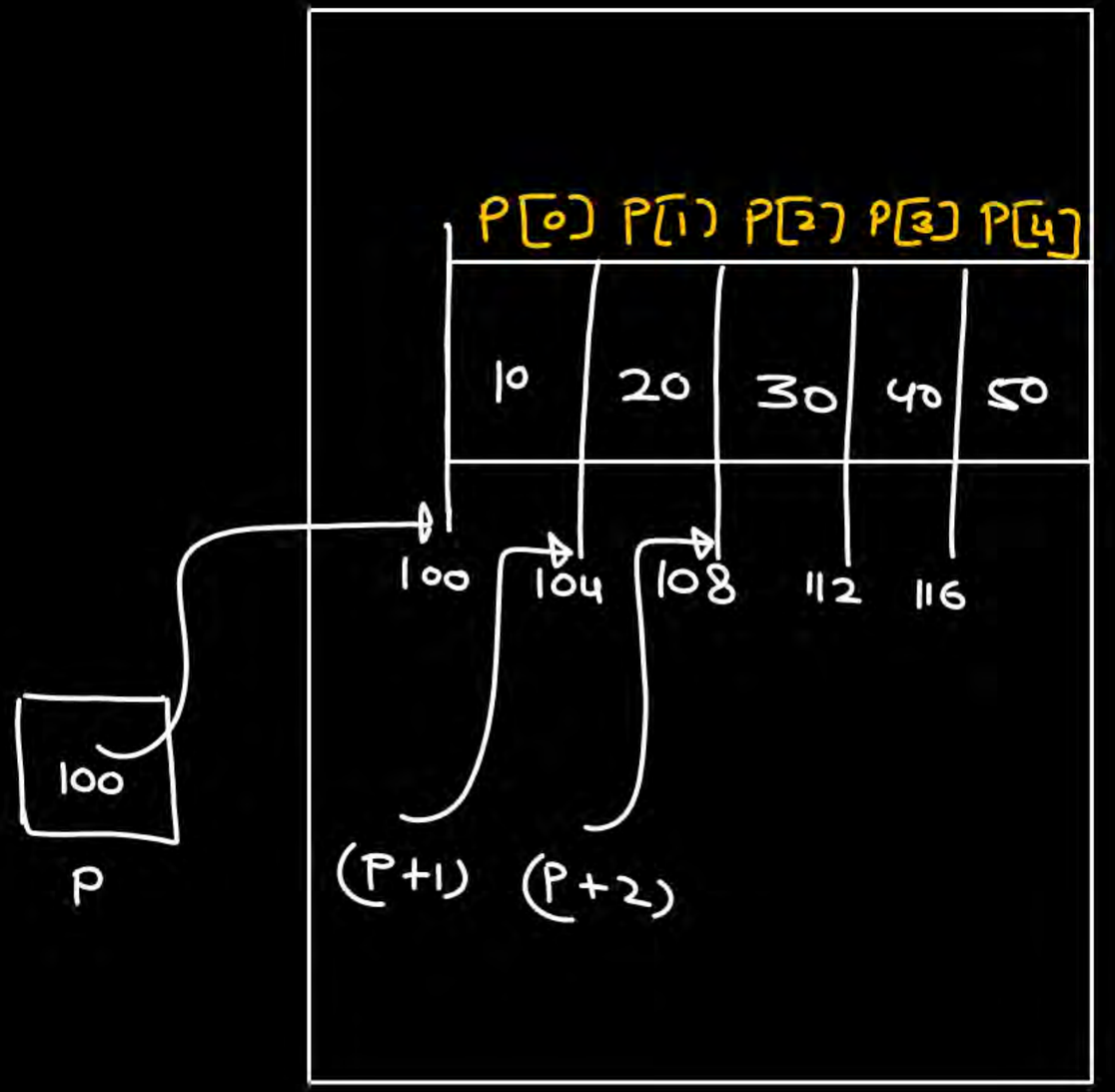


Sizeof

```
int *P;

P = malloc (5 x sizeof(int))

for(i=0; i<5; i++)
    sconf("%d", P+i);

for(i=0; i<5; i++)
    printf("%d", *(P+i));
```

P[i]

```
for(i=0; i<5; i++)
    pf("%d", P[i]);
```



```
void main(){
    int N, *P, i ;
    printf("Enter no. of elements");
    scanf("%d", &N);
    P = malloc( N x sizeof(int));
    if (P != NULL) {
        for(i=0 ; i< N ; i++)
            scanf("%d", P+i);
        for(i=0 ; i<N ; i++)
            printf("%d", P[i]);
    }
```

calloc

* work almost same as malloc

calloc(no. of block , size of each block)

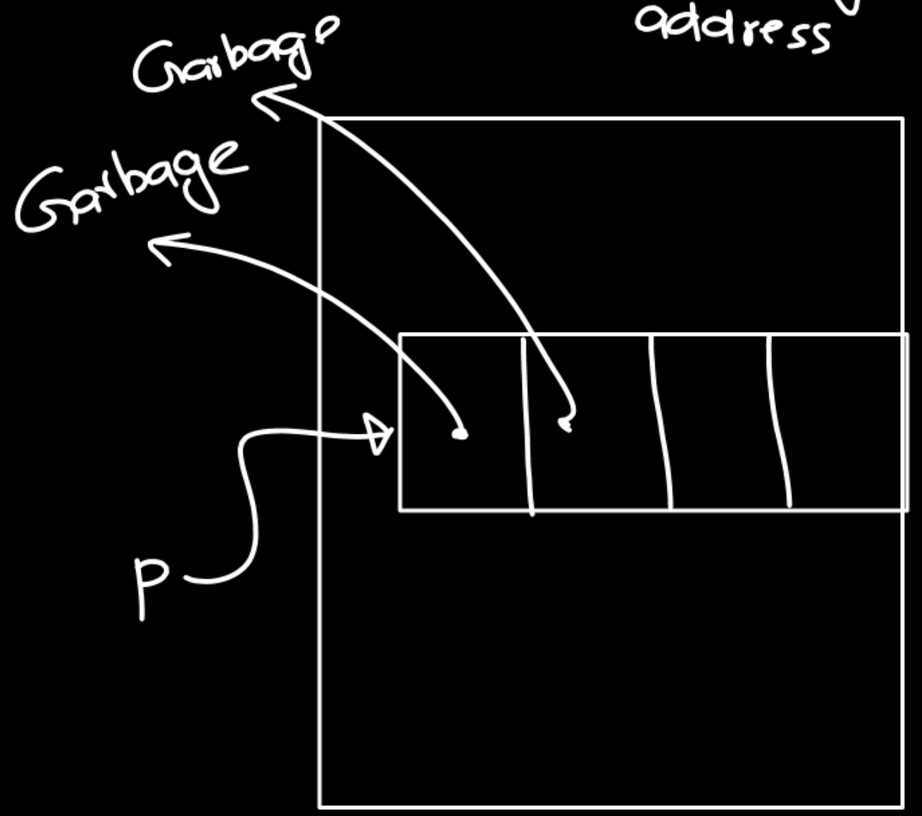
malloc(5 x sizeof(int))

calloc(5, sizeof(int));

malloc

1) Search

2) block is avail. → starting address



calloc

1) Search

2) block is avail

↳ bits ⟹ 0

3) starting add. return

[malloc is cheaper but not reliable
calloc is expensive but reliable]

08:30

i) realloc
ii) free

PYQ

array ✓

09:00 PM

1) strings - 1


THANK YOU GW
SOLDIERS !