CS & IT ENGINEERING

Programming in C

Arrays and Pointers

Lec- 08

By- Pankaj Sharma sir

- Dynamic Memory Allocation

```
void main(){
    int *p ;
    p++ ;
```

4byte

```
    char *p ;
    p++ ;
```

1 byte

```
void main(){
    void *p;
    int i = 10;
    Char ch = 'A';
    p = &i;
    printf("%d", *p);
```

→ derefrence Error

Solution:

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

→ typecasting

(i) Don't derefrence a void pointer without typecasting.

```
void main(){
    void *p;
    int i = 10;
    char ch = 'A';
    p = &i;
    printf("%d", *p);
```

→ dereference Error

Solution:

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

→ typecasting

```
    p = &ch;
    printf("%c", *p);
```

→ Error

```
    printf("%c", *(char*)p);
}
```

(i) Don't derefrence a void pointer without typecasting.

2:

```
void main(){
    void *p;
    int i = 10;
    p = &i;
    p = p+1;   ✗
    p++;       ✗
```

(i) Don't perform arith. operations on void pointer.

NULL Pointer

* Specially designed pointer.

valid pointer ⇒ that points to some valid address

Unsigned int

H. No -112 → -ve ✗

H. No. 10.563
K. N - Mathura → ✗

-ve no.

DS

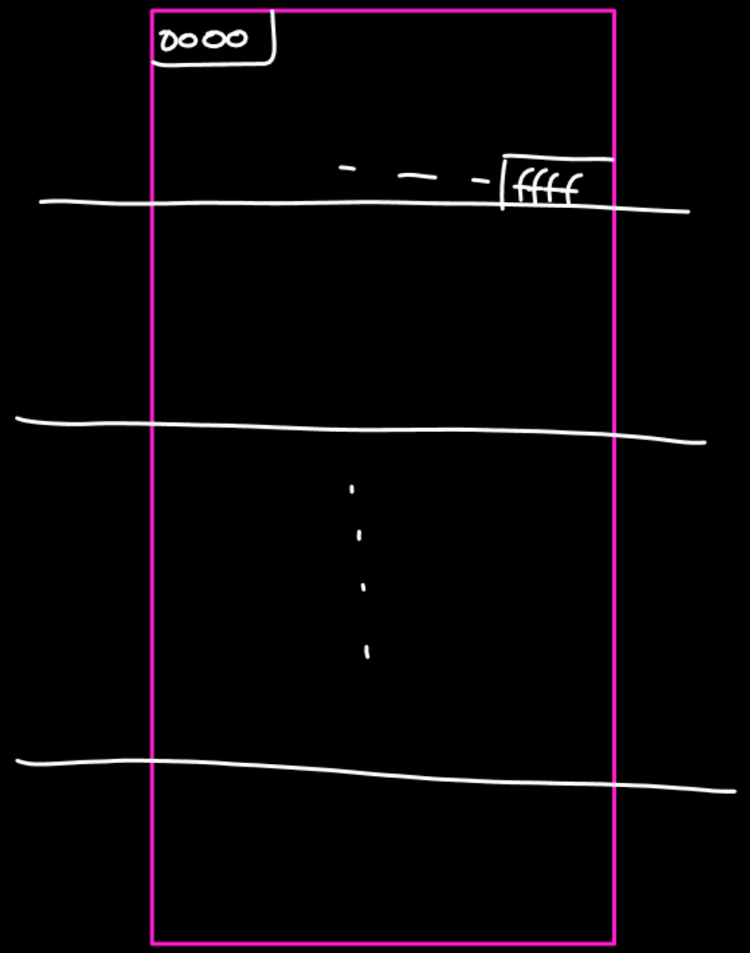
NULL → 0 ✓

(i) OS →

Advantage

error

false



Wild Pointer

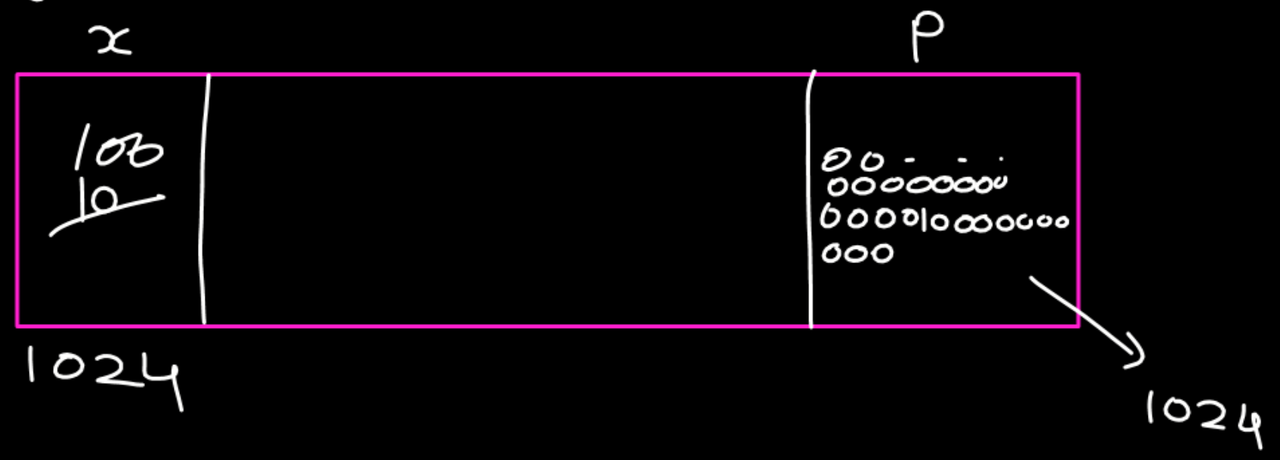
* Any uninitialized pointer

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

Garbage (some value — but not known to us)
(bits value - last)

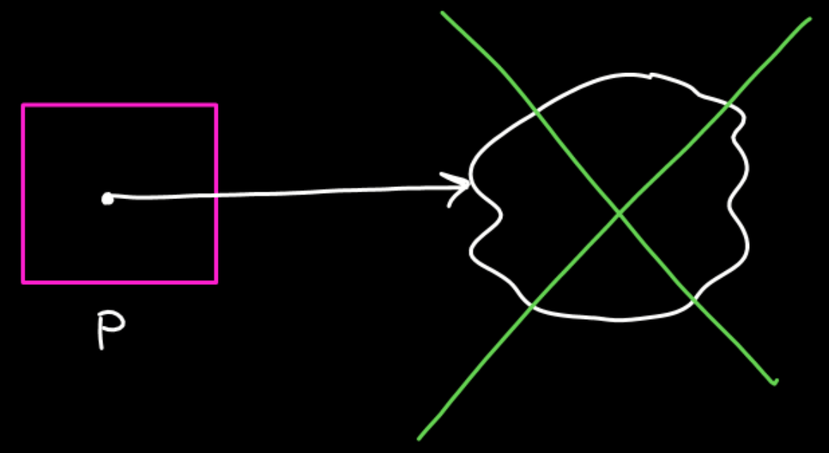
```
void main(){
    int *p;

    int x=10;

    *p=100;
```

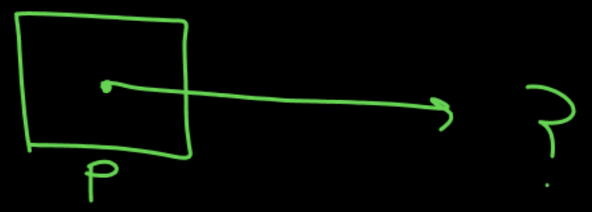
Garbage (some value)



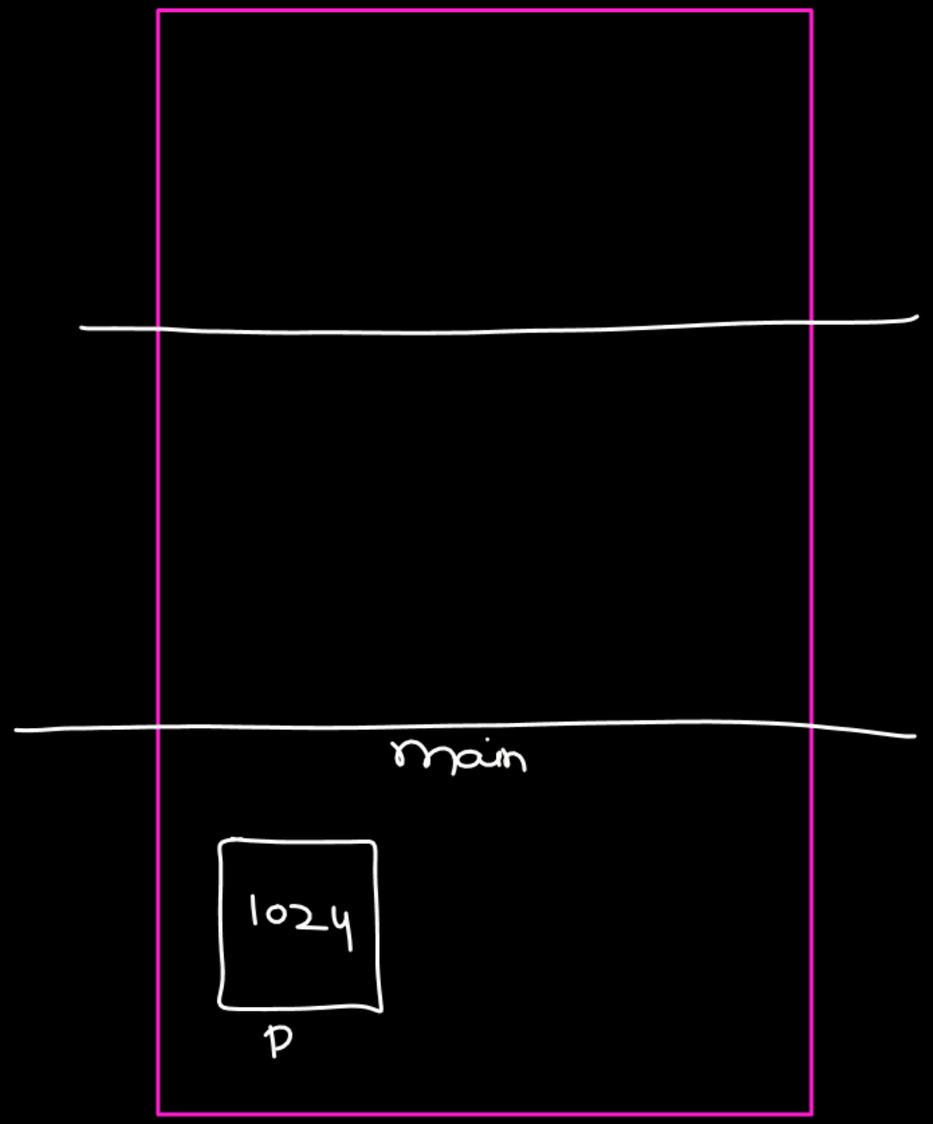
```
int *p = (int*)0;
```

Dangling Pointer





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

int * fun(){
        int x = 10 ;
        int *ptr ;
        Ptr = &x ;
        return Ptr ;
        }
```



① don't try to return add. of local var.

Dangling Pointer

✓, ✓, Error

```
int *f(){
    static int a=10;
    return &a;
}
void main(){
    int *p;
    p = f();
    pf("%d", *p);
}
```

Dynamic Memory Allocation

int A[ ] ;

Case 1 : int A[4000] ;

→ 500

memory wastage

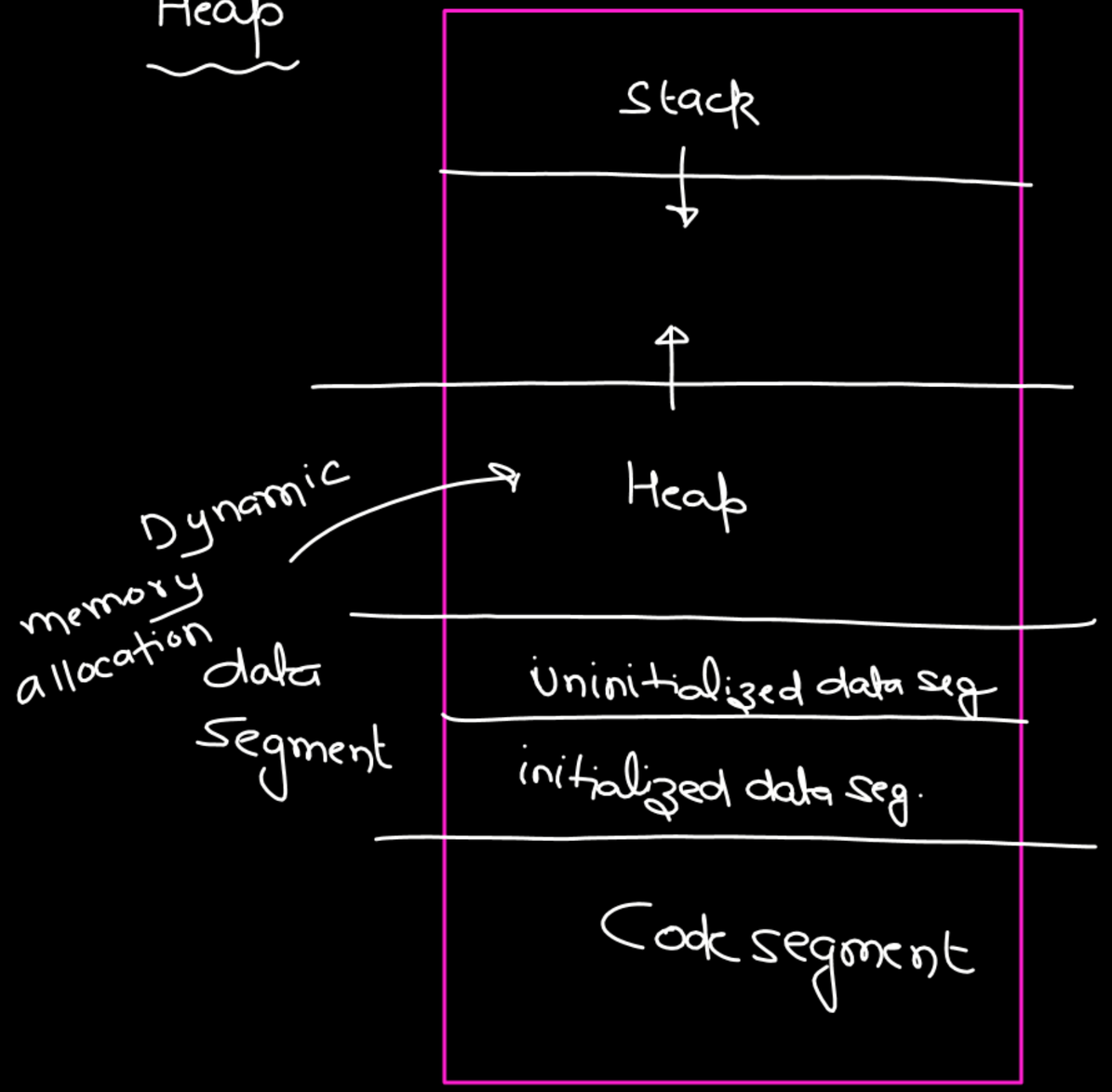
Case 2 : int A[500] ;

→ 4000

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

Little endian
Big Endian

Heap



malloc()

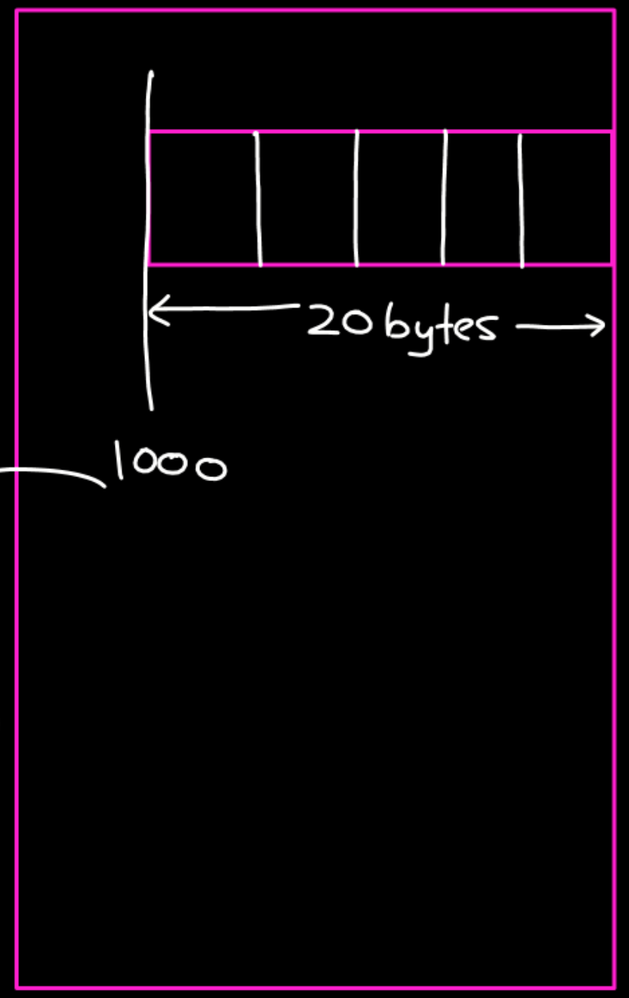
malloc(size in bytes)

malloc(20);

malloc

(void *) malloc(20)

Pointer ⇐ starting Address of the block



Heap

```
(void*) malloc(unsigned int);
```

int → 2 byte

stu1

store 5 integer

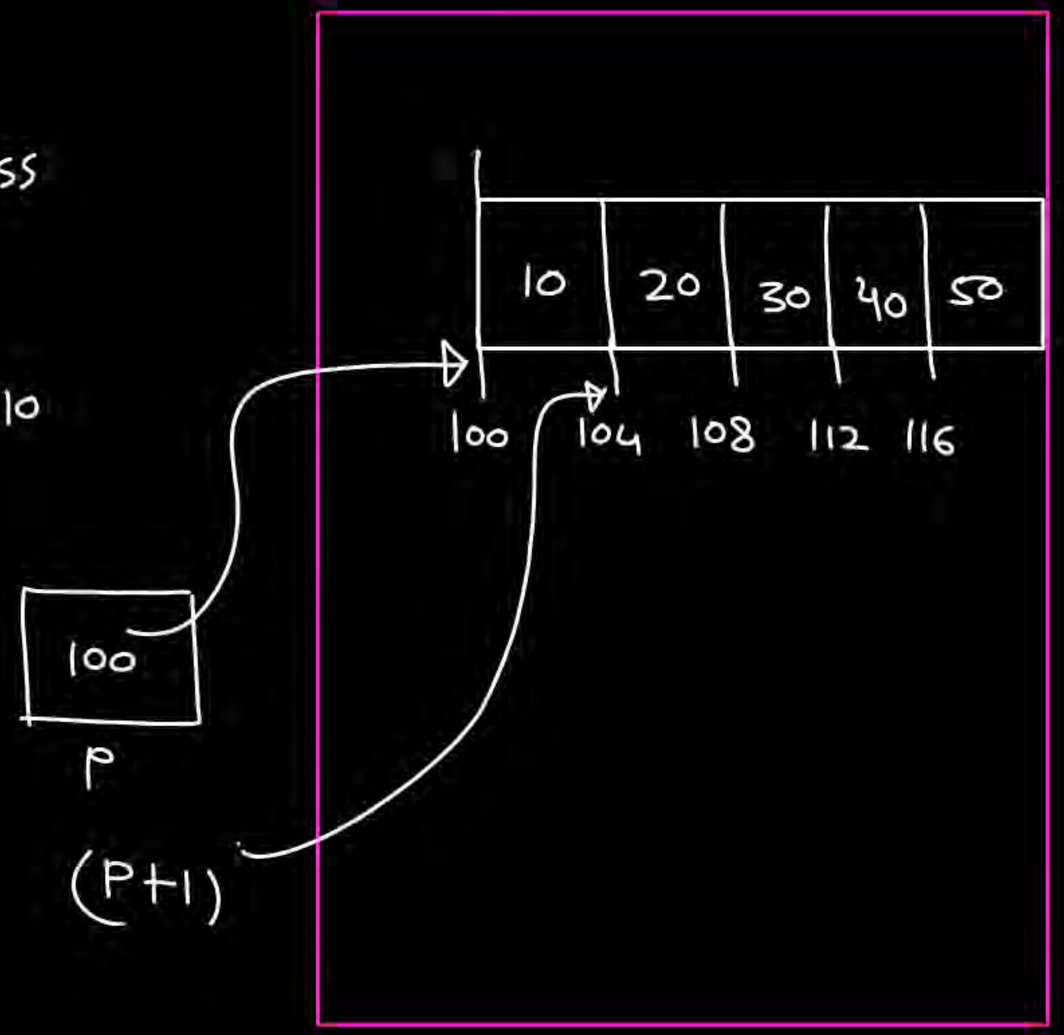
```
void main(){
    int *p;
    p = malloc(10);
    ≡
}
```

int → 4 byte

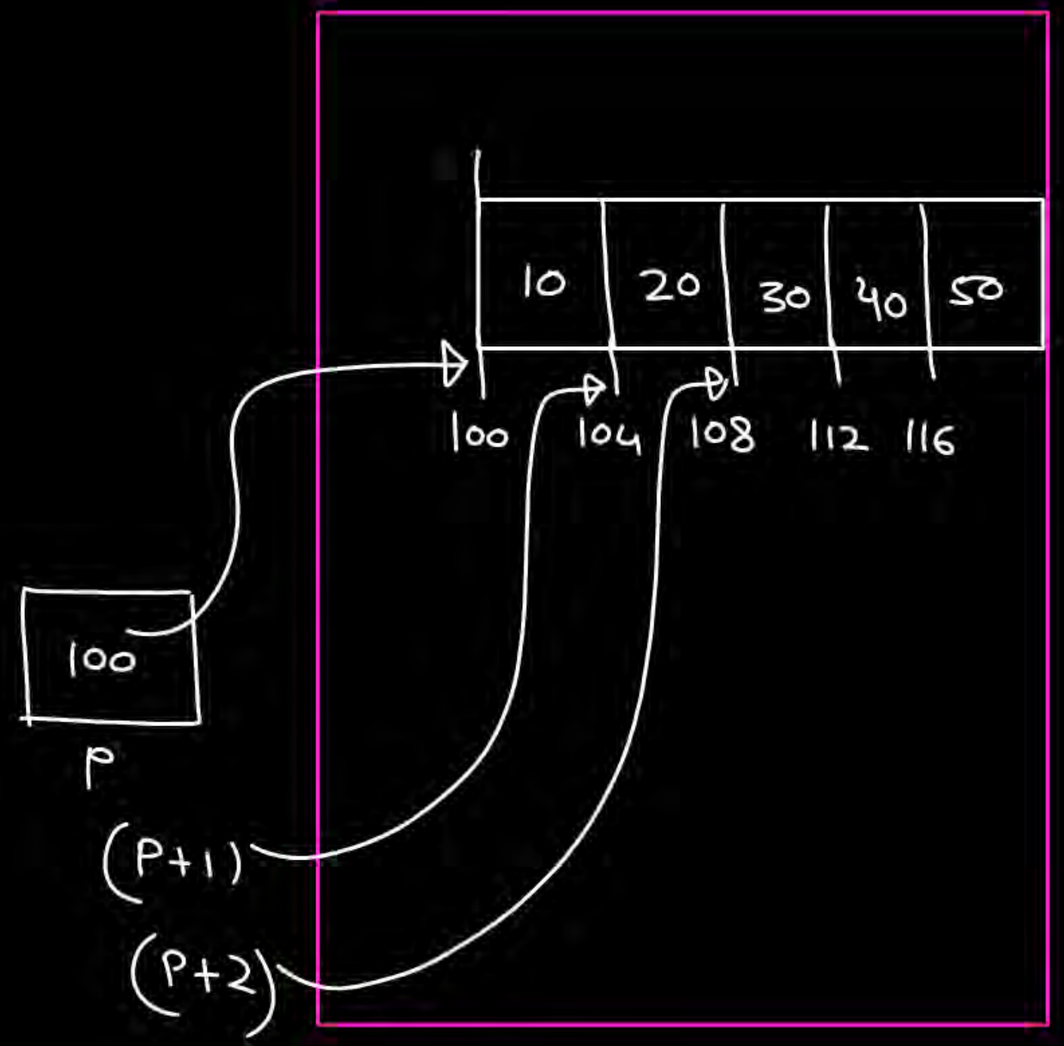
```
void main(){
    int *p;
    p = malloc(20);
    ≡
}
```

```
void main() {
    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
```

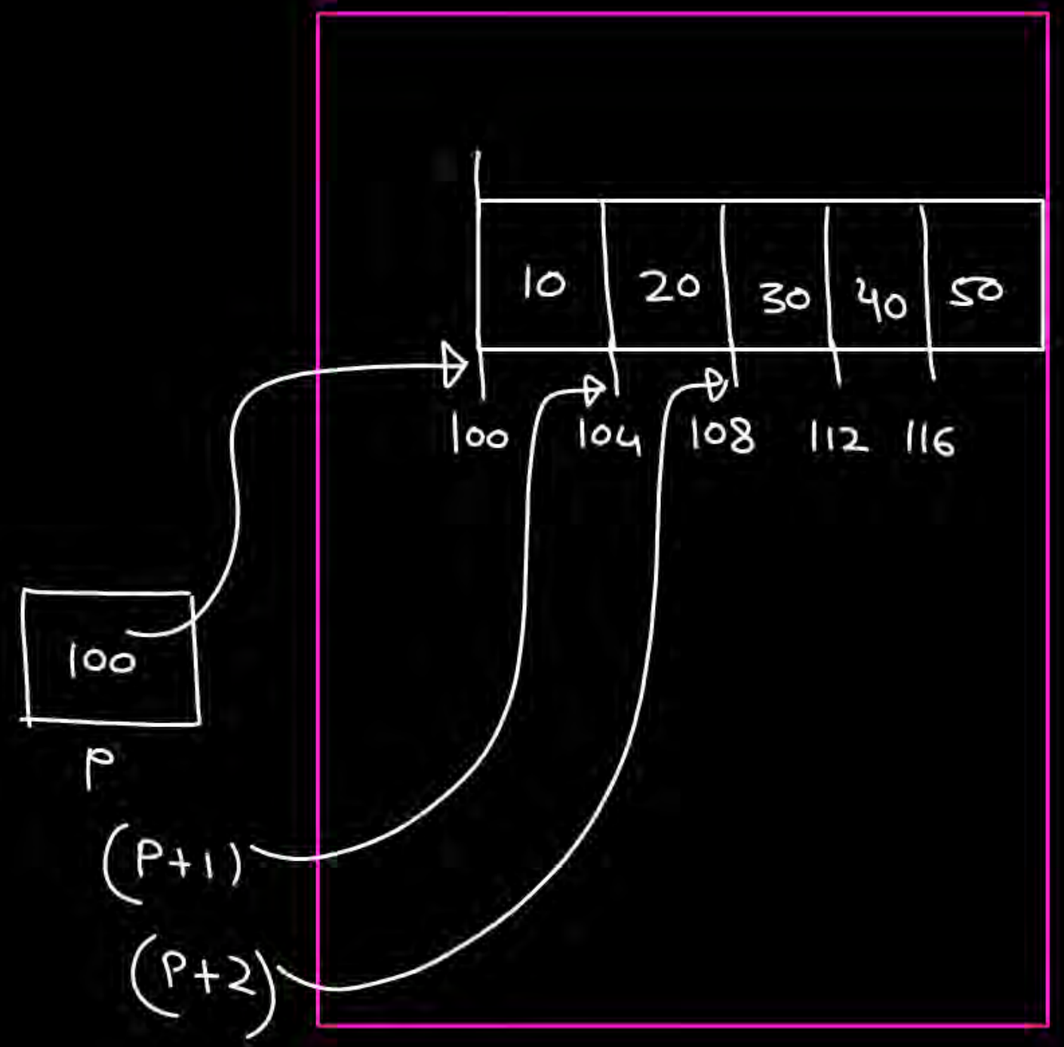
Address



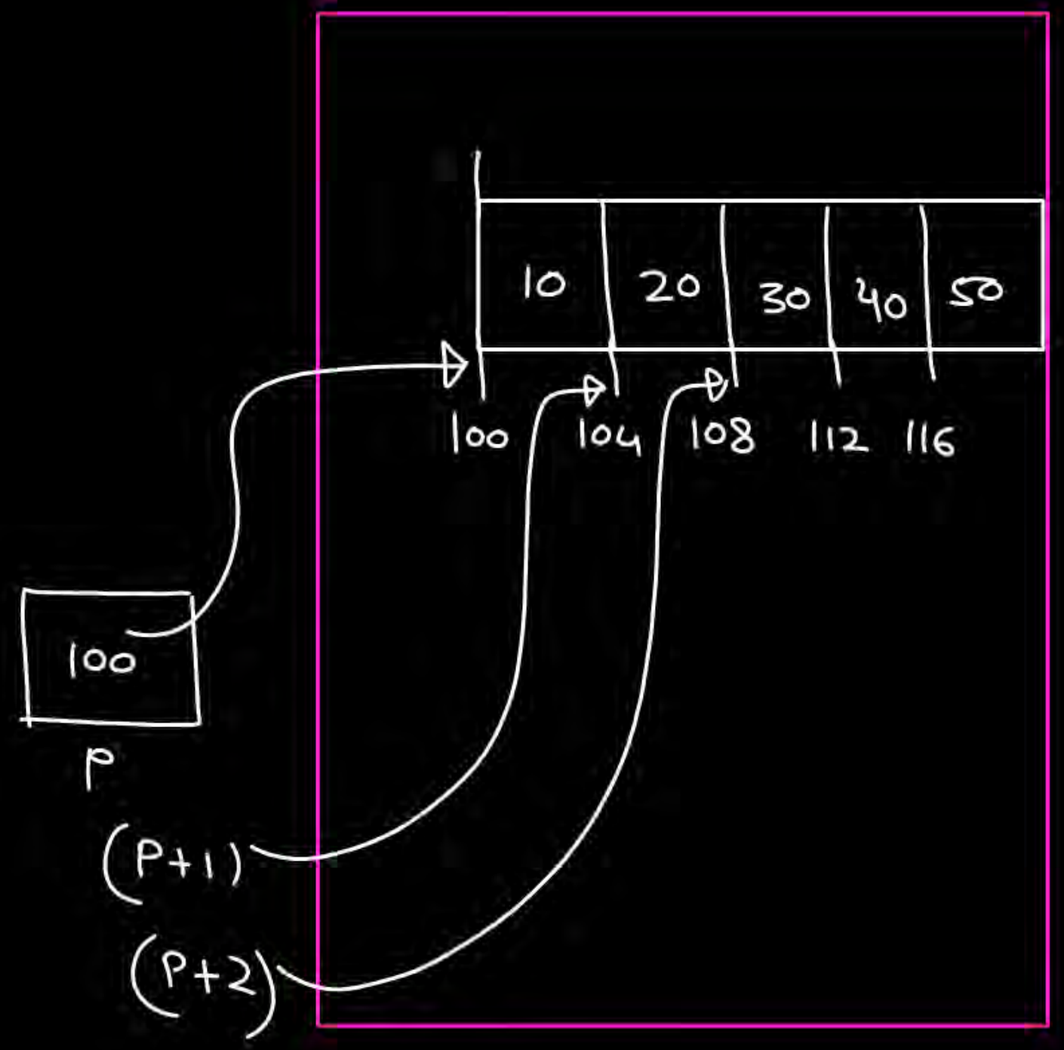
```
void main(){
    int *p ; int i ;
    P= malloc( 5 x sizeof(int));
    for(i=0;i<5;i++)
        Scanf("%d", P+ i) ;

    printf("%d", *p); ⇒ 10
        pf("%d", *(p+1)); ⇒ 20
        pf("%d", *(p+2)); 30
        pf("%d", *(p+3)); 46
        pf("%d", *(p+4)); 50
```



```
void main(){
    int *p ; int i ;
    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));

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



```
void main(){
    int *p ; int i ;
    P= malloc( 5 x sizeof(int));
    for(i=0;i<5;i++)
        Scanf("%d", P+ i) ;
    for(i=0;i<5;i++)
        pf("%d", P[i]);
    }
```



malloc(10000)

Available

10,000 bytes

↓ block-search

starting address of the block u demanded

Not available

Can it return any address

NULL

5 int ✓

```
void main() {
    int *p;
    int N, i;
  printf("Enter no. of elements");
  scanf("%d", &N);
    p = malloc(N X sizeof(int));
```

valid
odd

NULL

```
    if(p != NULL) or if(p)
  {
      for(i=0; i<N; i++)
            scanf("%d", p+i);

      for(i=0; i<N; i++)
            printf("%d", p[i]);
    }
}
```

calloc

* work same as malloc

How they differ

calloc(No. of blocks , size of each block)

malloc(5 x sizeof(int));

(void*) calloc(5, sizeof(int));

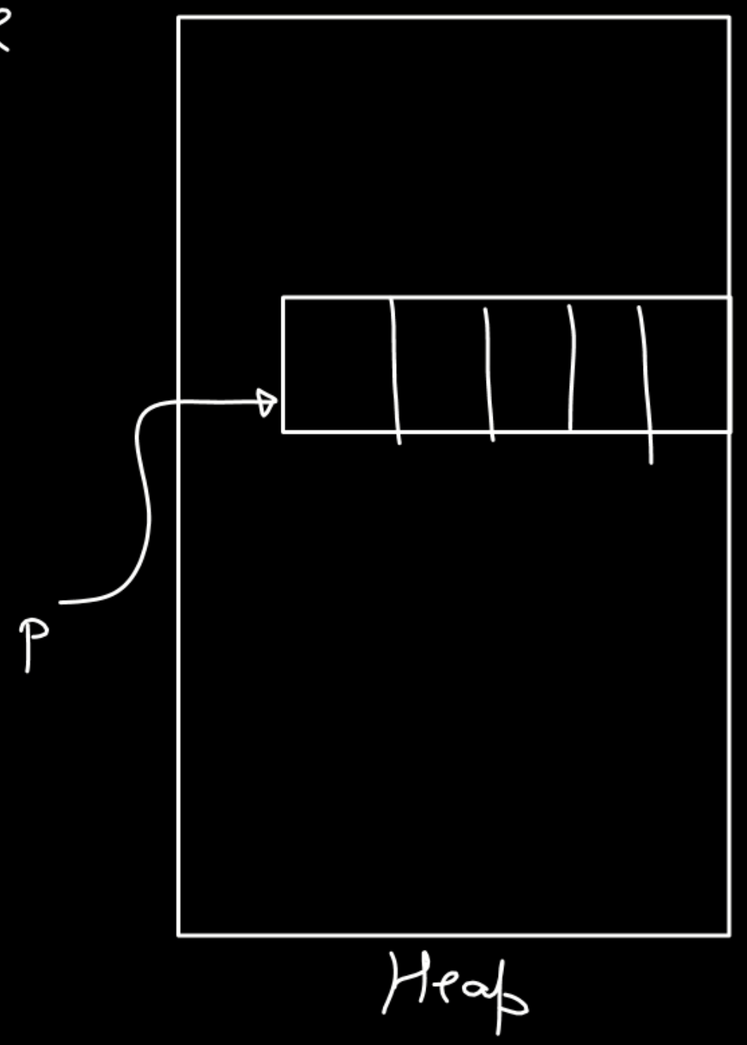
malloc(5* sizeof(int))

→ search

→ Available → starting add. return

Garbage

printf("%d", *p)



calloc(5, sizeof(int));

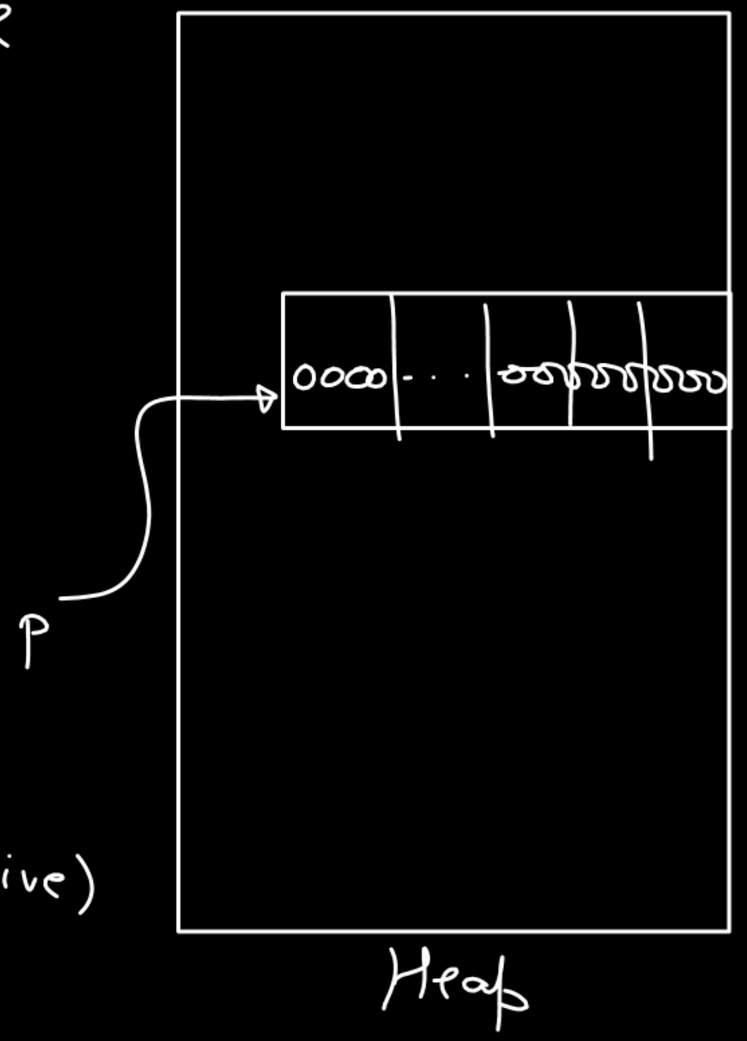
Garbage

→ search

→ सारे bits → 0

→ starting address return

printf("%d", *p)



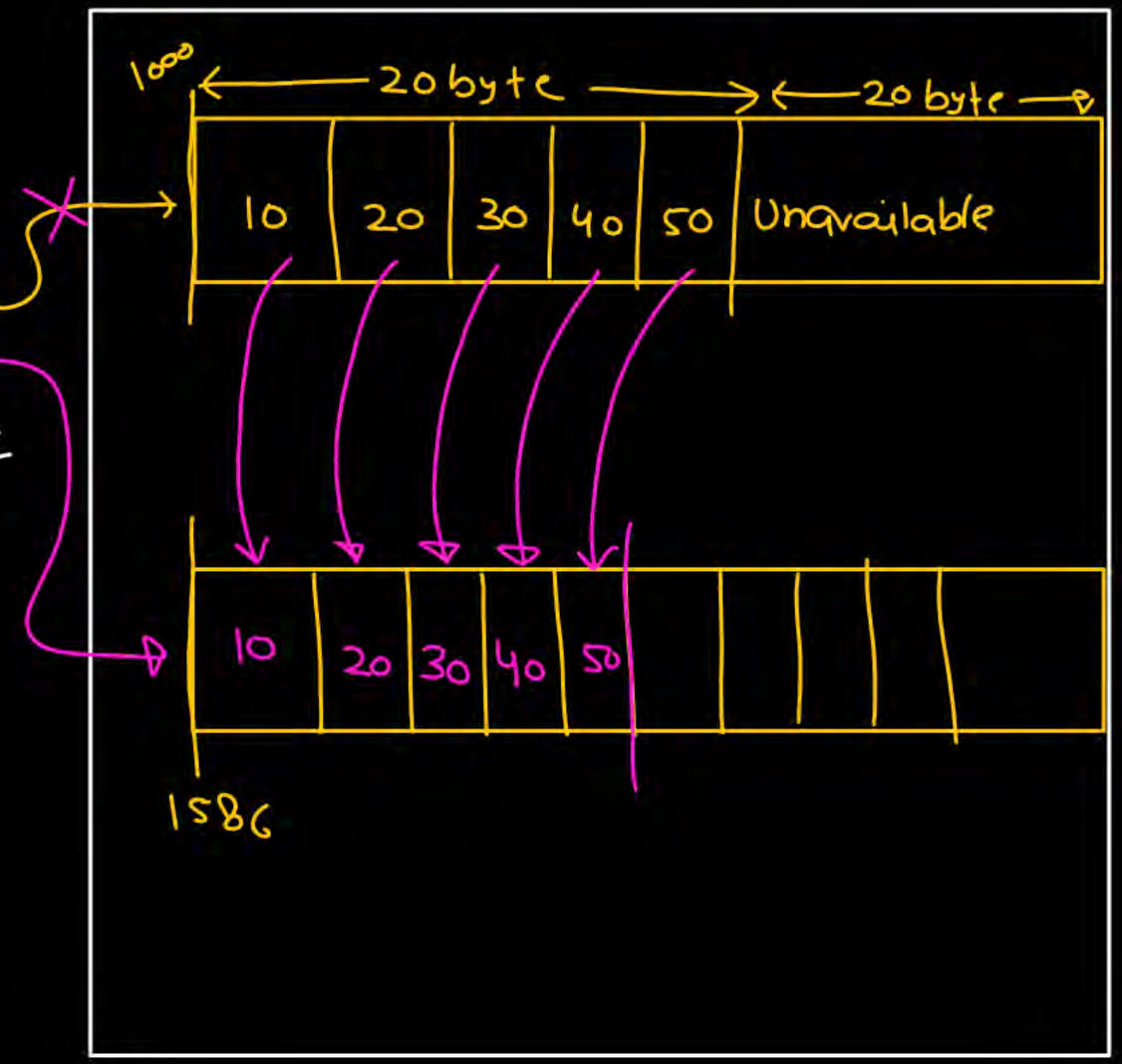
malloc	calloc
less time (cheaper)	more time (expensive)
Not Reliable	Reliable

realloc

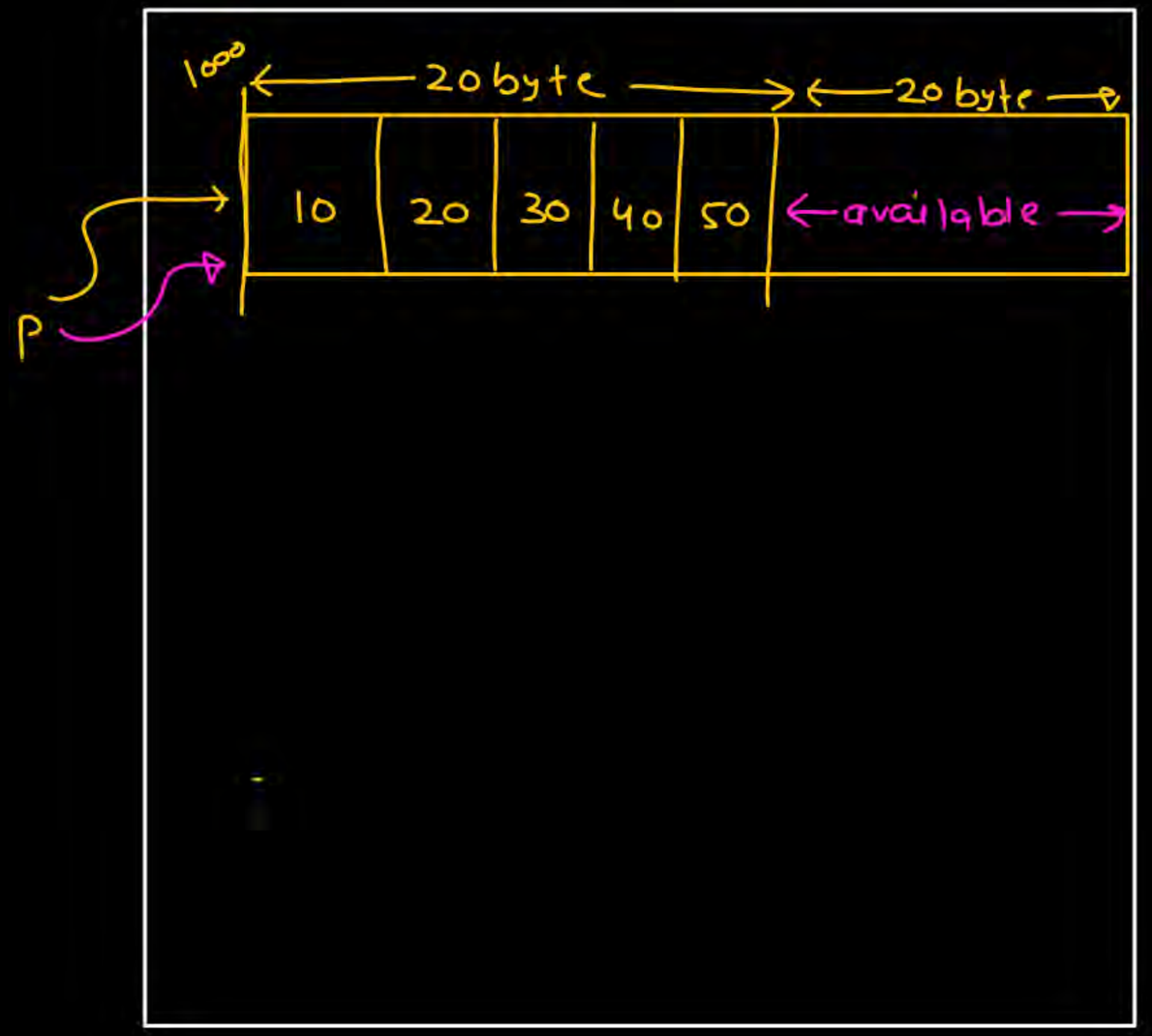
```
int *p = malloc(sizeof(int) × 5);
    for(      )
        scanf - -
```

prev. memory is allocated either by using malloc or calloc

```
int *p = malloc(sizeof(int) × 5);
    for(     )
        scanf - -

p = realloc( p , sizeof(int)×10);
```

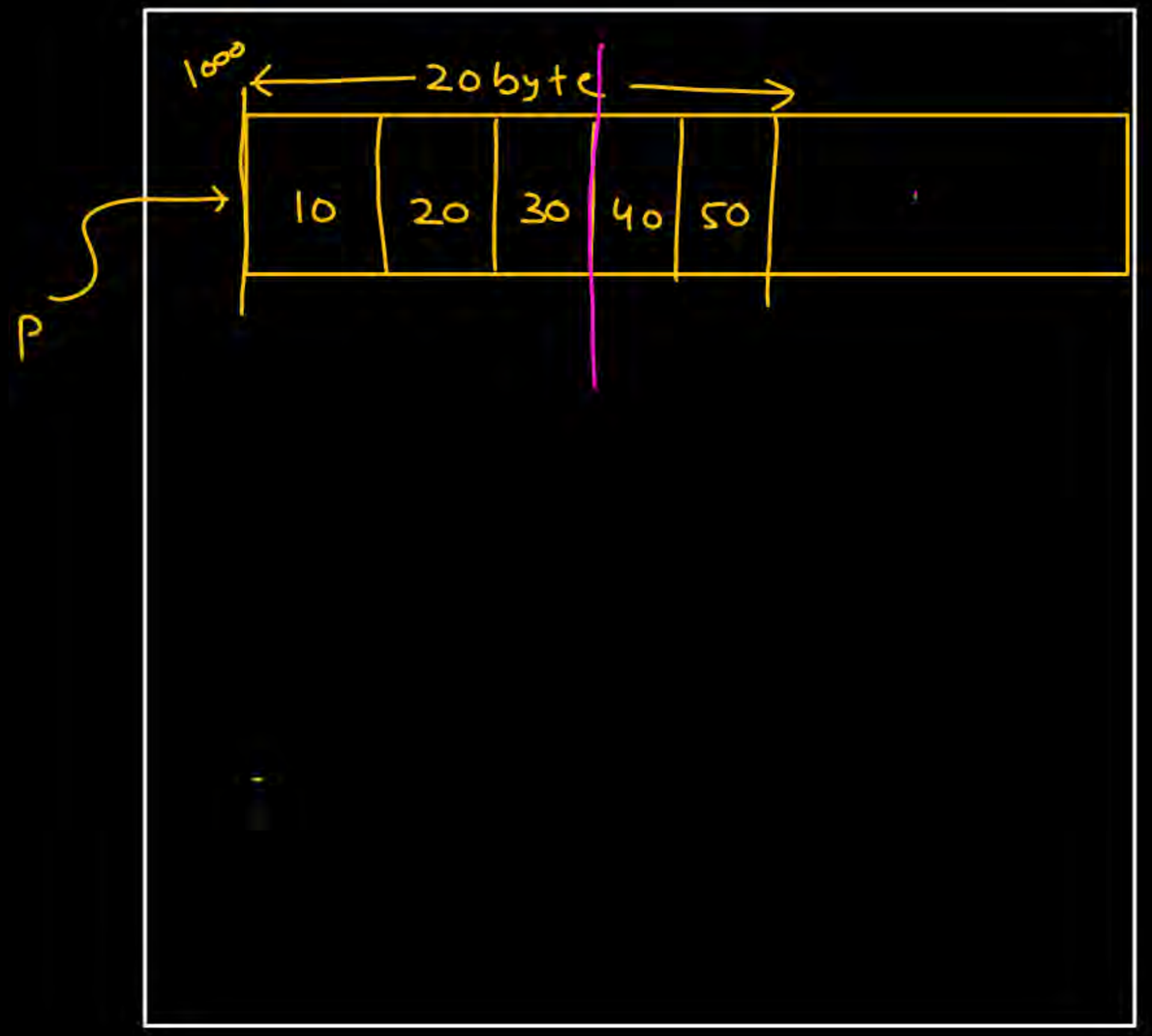


```
int *p = malloc(sizeof(int) x 5);
    for(     )
        scanf - -

P = realloc( P , sizeof(int) x 10);
```



```
int *p = malloc(sizeof(int) × 5);
    for(     )
        scanf - -

P = realloc( P , sizeof(int)×3);
```

HW




THANK YOU GW
SOLDIERS !