17 Structure Pointer-

```
#include<stdio.h>
void main()
{
    struct book_name{
            char name[30];
            int page;
            int cost;
          };
typedef struct book_name book;
book b1={"Let us c",430,240};
book *p;
p=&b1;

printf("%s%d%d\n",b1.name,b1.page,b1.cost);
printf("%s%d%d\n",p->name,p->page,p->cost);
}
```

Structure variable. ← b1

p → pointer of structure type

p → pointer to structure

To access members of structure with structure variable, we use dot . operator, but when we have a pointer of structure type, we use arrow → to access structure members.

Output → Let us C 430 240
Let us C 430 240

27 Dynamic Memory Allocation-

int data[500];

It allocates an array of int of size 500, but if we don't know the required no. of data then we can allocate memory dynamically.

Dynamic memory management refers to manual memory management. This allows to obtain more memory when required and release it when not necessary. For DMA, 4 library functions are defined under <stdlib.h>

malloc() -> allocates requested size of bytes and returns a pointer first byte of allocated space.

calloc() -> allocates space for an array elements, initializes to zero and then return a pointer to memory.

free() -> deallocate the previously allocated space.

realloc() -> change the size of previously allocated space.

realloc (p, n)
- p → Pointer to prev allocation
- n → new size

⇒ malloc stands for memory allocation. This function reserves a block of memory of specified size and returns a pointer of type void which can be casted into pointer of any form.

(cast type *) malloc(byte-size)

If the space is insufficient, allocation fails and returns NULL pointer.

eg- int *x;
x = (int *) malloc(100 * sizeof(int));

memory space allocated to variable x.

This statement will allocate either 200 or 400 bytes and the pointer points to address of first byte of memory

⇒ Suppose we want an array of size n -

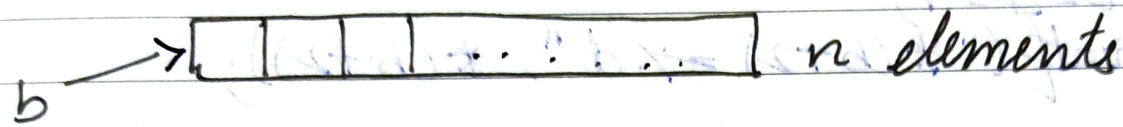
First read the value of n by scanf(). We will define a variable whose contain will be starting address of that array. We will define b a pointer to an int. Then we will allocate n elements of that size.

int *b;
b = (int *) malloc(n * sizeof(int));

type casting.

Why type casting is required -

" malloc (n * size of (int))" actually used to allocate (n * size) memory and then starting address will come but we need to assign the starting address into b and type of b is int type pointer.



=> calloc stands for contiguous allocation.

(cast-type*) calloc (n, element-size);

This statement will allocate contiguous space in memory for an array of n elements.

eg. x = (float *) calloc (25, sizeof (float));

allocates contiguous space in memory for an array of 25 elements each of 4 bytes.

=>

calloc()	malloc()
1) allocates multiple blocks of memory each of same size.	1) allocates single block of memory.
2) initializes the allocated memory with 0 value.	2) initializes the allocated memory with garbage values.
3) (cast type *) calloc (blocks, size of blocks);	3) (cast type *) malloc (size in byte
4) No of arguments 2	4) No. of arguments 1

Eg.
```
void main()
{
    int *b;
    b=(int *) malloc (10 * sizeof(int));
    printf("%u is b", b);
    printf("\n(b+1) = %u", (b+1));
    printf("\n(b+2) = %u", (b+2));
    b[2]=4;
    printf("*(b+2) = %d", *(b+2));
```

Output -

556004 is b
(b+1) = 556006
(b+2) = 556008
*(b+2) = 4

Eg: Allocate 2D Array -

```
main()
{
  int *a[5], i;
  for(i=0; i<5; i++)
    a[i]=(int *)malloc (10 * sizeof(int));
```