```
int i;
for(i= 0; i<4 ;i++)
    Printf ("%u\n", s[i]);
}
```

O/P:

```
65508
65516
65524
65532
```

s[2] → 65524 → address of 2nd 1-D array

s[2][1]

↳ (s[2]+1) → 65524 + 1

→ 65528

* (s[2]+1).

⇓

$\boxed{*(*(s+2)+1).}$ ∵ num[i] = *(num+i)

Pointer to an array (Array pointer)

A pointer to an array is a pointer that points to the whole array instead of the 1st element of the array.

example.

It considers the whole array as a single unit instead of it being a collection of elements.

syntax: type (* Ptr)[size];

ex) int (* ptr)[10];

```
main()
int a[3]= {1,2,3};
 int * ptr= a;
             points to
             the
             base addr
Printf("%d\n", *ptr);
```

```
main()
int a[3]= {1,2,3}
   int (* ptr)[3];
            ptr= a;
for (int i=0; i<3; i++)
     Printf(" %d", (*ptr)[i]
}.
```

Array of pointers

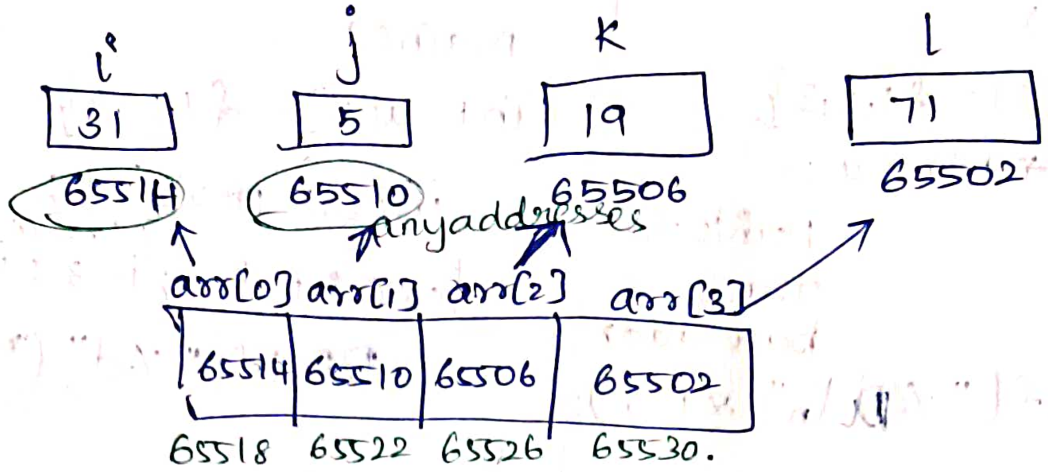
array of ints | floats.

Array of ptrs → collection of addresses.

Addresses → can be address of any variable, address of any array or random address.

Rules of array → apply to array of pointers.

```
#include <stdio.h>
main()
{
    int *arr[4];
     int i= 31, j=5, k=19, l=71, m;
       arr[0]= &i;
       arr[1] = &j;
       arr[2] = &k;
       arr[3] = &l;
     for (m=0; m<=3; m++)
    {
           Printf("%d\n", *(arr[m]));
}
```



Strings

```
#include <stdio.h>
main()
{
    char name[] = "Klinsman";
    char *ptr;
    ptr = name;  → stores the base
                   address
    while (*ptr != '\0')
    {
        printf("%c", *ptr);
        ptr++;
    }
}
```

→ incrementing a pointer, it points to the immediately next loc. of its type

$name[i] = *(name+i) = *(i+name)$

$= i[name]$.

ex

```
char str1[] = "Hello";
str2[] = "Hi";
char *p = "Hello", char *q = "Hi";
```

either store as char (array) / store in some memory loc and assign the address to the char ptr.

you cannot change the ptr but change value pointed by ptr.

Str1 → constant pointer to a string

P → pointer to a constant string. (you cannot change the value pointed by ptr, but you can change the ptr itself)

Str1 = "Adieu"   X, → const. pointer cannot change

str1 = str2   X →   "

str1++   X →   "

* str1 = 'z'   works, because string is not constant.

P = "Adieu"   works, because pointer is not constant.

P = s ,   works "

p++ ,   works "

* p = 'M'   error, because string is constant.

-- - -

Program to detmn the length of a char string

```
main()
{
    char *name;
    int length;
    char *cptr = name;
    name = "DELHI";
    printf ("%s\n", name);
    while (*cptr != '\0')
    {
        Printf ( "%c is stored at
                     address %u", *cptr, cptr);
    }   cptr++;
    length = cptr - name;
    printf("Length of the string = %d\n", length);
}.
```

o/p:
D is stored at address 5000

Length of the string = 5.

Dynamic memory allocation

for allocating memory dynamically,

1) malloc() → 1 parameters (size)
2) calloc()
3) realloc()
4) free().

} all in <stdlib.h>

Static: A variable defined in a func, is stored in stack memory. It needs to know the size of the data to memory at compile time. (before the program runs).

Also once defined, we can neither change the size nor completely delete the memory.

↓ to overcome this, (DMA) → allows you to allocate memory at runtime, giving the ability to handle data of varying sizes.

Dynamic resources → in heap memory.

ex) array: if size not sufficient → set to ~~store~~ allocate the maximum size possible