```
Scanf("%d", &ndisk);
printf("sequence is: ");
ToH(ndisk, source, temp, dest)
}
```

→ Arrays
- → 1-D array
- → 2-D array
- → multi-dimensional array

1-D Array

Declaration of 1-D array

datatype arrayname [size];

```
int   Emped[100];
float Salary[100];
```

Accessing 1-D array elements

```
int arr[5];
```

In C, array subscripts start from 0.

arr[0].

Processing 1-D arrays

Program to input values in an array and display them

```
#include <stdio.h>

main()
{
   Int arr[5], i;
for (i=0; i<5; i++)
 {
   printf("Enter the value of arr[%d]: ", i);
   scanf("%d", &arr[i]);
}
```

```
printf("The array elements are : \n");
for (i=0; i<5; i++)
{
    printf("%d", arr[i]);
}
}
```

→ Program to add the elements of an array

```
#include <stdio.h>
main()
{
    int a[5], i, sum=0;
    for (i=0, i<5, i++)
    {
        printf("Enter the value of a[%d]", i);
        scanf("%d", &a[i]);

        sum += a[i];
    }
    printf("The sum of array elements = %d", sum);
```

→ Program to count the even and odd number in an array.

```
#include <stdio.h>
main()
{
    int arr[10], i, even=0, odd=0;

    for(i=0 ; i<10 ; i++)
    printf("Enter the values of arr[%d] : ", i);
    scanf("%d", &arr[i]);

    if (arr[i] % 2 == 0)
        even++;                    printf("Even nos = %d,
    else                                odd nos = %d", even,
        odd++;                              odd);
```

Initialization of 1-D Array

datatype arrayname[size] = {value 1, value 2, ... value N};

Ex: int arr[5] = {10, 20, 30, 40, 50};

int marks[] = {91, 92, 93, 94, 100};

arr[5] = {1, 2, 3, 4, 5, 6, 7, 8} → Error

→ Program to find minimum and maximum number in an array.

```
#includes <stdio.h>
main()
{
    int arr[5] = {1, 2, 3, 4, 5};
    int i, num, max;
    min = max = arr[0];

    for (i=0; i<=4; i++)
    {
        if (arr[i] < min
            min = arr[i];
        if (arr[i] > max
            max = arr[i];
    }
```

→ 1-D Arrays and Functions

Passing 1-D array elements to a function.

```
#include <stdio.h>
    main()
{
    int arr[5], i;
    printf("Enter array element s ");
```

```
for (i=0, i<5, i++)
scanf ("%d", &arr[i]);
Check (arr[i]);

check (int num)
{
   if (num%2 == 0)
   printf ("Even no: ");
    else
   printf ("odd no: ");
}
```

→ Passing whole 1-D array to a function:

```
# include <stdio.h>
  main()

{ int arr[5] = {1,2,3,4,5}, i;
     fun(arr);

  printf ("Content of the array now: ");
   for (i=0; i<5, i++)
   {
    printf ("%d", arr[i]);
    }
  fun (int val[])
  {
   int sum=0, i;
  {  val[i] = val[i] x val[i];
     sum += val[i];
  }
  printf ("The sum of square = %d", sum);
}
```

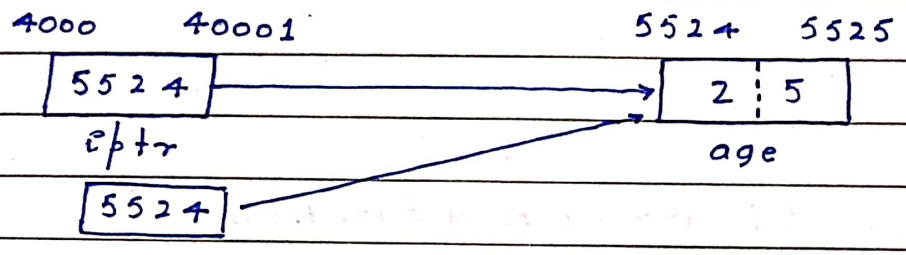
→ Pointers :

It is a special variable that stores the address of another variable.

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

Base type: int ↘ address of

```
int *pptr, age = 25;
iptr = &age;
```




```
iptr = NULL;
```

We can access the value of a variable indirectly too.

```
int a = 87;
float b = 4.5;
int *p1 = &a;
float *p2 = &b;
```

*p1 = 9 is equivalent to a=9

(*p1)++ " " a++

x = *p2 + 10 " x = b + 10

printf("%d %f", *p1, *p2) is equivalent to printf("%d %f", a, b)

printf("%d %f", p1, p2) is equivalent to printf("%d %f", &a, &b)

* → indirection operator

↳ value at address

Dynamic Memory Allocation

If we want to allocate memory dynamically at runtime.
It is allocated from heap memory.

malloc(), calloc(), realloc(),
These functions are used to allocate the memory dynamically at runtime.

Malloc(): defined in stdlib.h; specifies the no. of bytes

Declaration: `void * malloc (size_t size);`

returns a pointer to the first byte of memory allocated

func() used to allocate memory dynamically

It is generally used as:

```
ptr = (datatype*) malloc (specified size);
```
↳ pointer of type datatype

For ex: `int * ptr`

```
ptr = (int *) malloc (10);
```

If we want to make our program portable and readable

```
ptr = (int *) malloc (5 * sizeof (int));
```

```
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>

main ()
{
  int * P, i, n;
  printf ("Enter the no. of integers to be entered: ");
  scanf ("%d", &n);
  p = (int *) malloc (n * size of (int));
  if (p == NULL)
  {
```

```
printf("Memory not allocated");
exit(1);
}

for (i=0; i<n; i++)
{
printf("Enter an integer");
scanf("%d", p+i);
}
for (i=0; i<n; i++)
{
printf("%d", *(p+i));
}
}
```

Calloc()

Declaration: void* calloc(size_t n, size_t size);
(n: No. of Block; size: size of each block)

allocates multiple blocks of memory. It takes two arguments.

Ex:
```
ptr = (int*)calloc
(5, sizeof(int));
```

→ Realloc()

Declaration: void* realloc(void *ptr, size_t newsize);

Ex:
```
ptr = (int*)malloc(size);
ptr = (int*)realloc(ptr, newsize);
```

→ Free()

Declaration: void* free(void*ptr)

used to free or deallocate the memory space allocated
The memory released is made available to heap again

```
free(ptr);
```