⊙ Arrays :

↳ an array is a collection of elements of the same data type, stored in a contiguous memory locations.

↳ array declaration - `int numbers[5];` // an integer arr with 5 elements

↳ array initialization - `int numbers[5] = {10, 20, 30, 40, 50};`

or

`int numbers[] = {1, 2, 3, 4, 5};`

Q-1 : Array Intro

a) why do we need arrays ?

↳ without arrays, if we need to store 100 numbers, we must declare 100 variables.

↳ int num1, num2 - - - numbers.

b) advantages of array

↳ allowing storing multiple values of the same type using single variable

↳ reduces memory wastage

↳ enables fast access to elements using indexing.

↳ promotes sorting and searching algorithms.

c) disadvantages of array :

↳ fixed size, cannot resize after declaration

↳ Memory wastage : if allocated more space than req.

d) Types of arrays :

(a) one-dimensional array -

↳ a 1-d array is a list of elements stored in a contiguous memory location.

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};      // declr. with initialization

    // accessing array elements
    for(int i = 0; i<5 ; i++){
        printf("%d", arr[i]);
    }                                        // 10 20 30 40
    return 0;
}
```

(b) Two-dimensional Arrays

↳ an 2D array is an array of arrays. Often used to repres. matrices.

```
↳ #include <stdio.h>
    int main() {
        int matrix[2][3] = {{1,2,3}, {4,5,6}};
                                  // 2 Rows x 3 columns
        // accessing elements;
    for(int i = 0; i<=2; i++){
            for(int j = 0; j<3; j++){
                printf("%d", matrix[i][j]);
            }
            printf("\n");
        }
        return 0;
    }
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

GROOT ACADEMY

(c) Multidimensional Array

- this kind of array extends beyond 2D, such as 3D array.
- `int arr[2][2][2] = { {{1,2}, {3,4}}, {{5,6}, {7,8}} };`

~~`printf("%d", arr[1]`~~

(d) Dynamic Arrays

- in C, dynamic arrays use malloc() & calloc() for memory allocation.
- `#include <stdio.h>` → `#include <stdlib.h>`

```
int main() {
    int *arr;
    int n = 5;
    arr = (int *) malloc (n * sizeof(int));   // allocate mem.
    for (int i = 0; i < n; i++) {
        arr[i] = i+1;
    }
    for (int i = 0; i < n; i++) {
        printf("%d", arr[i]);
    }
    free(arr);
    return 0;
}
```

// free memory

1 2 3 4 5

allocate memory for 5 integers

assigning values

Printing values

freeing allocated memory

* They allocate mem. from heap so free them to prevent memory leaks

	malloc()	calloc()
Memory Blocks →	Single Block	Multiple Blocks
initialg. →	garbage values	initialized to 0
Param. →	size in bytes	num of elem., size.
speed →	faster	Slightly Slower

+91-8233266276
info@grootacademy.com
122/166, 2nd Floor, vijay path, Mansarovar, 302020
www.Grootacademy.com

GROOT ACADEMY

Array Operations:

a) Traversing an array
↳ looping through the array using a for loop

↳
```
for (int i = 0; i < n; i++) {
    printf("%d", arr[i]);
}
```

b) Insertion in an array
↳
```
#include <stdio.h>
int main () {
    int arr[6] = {1,2,3,4,5,6};   // 5 elements only
    int pos = 2, num = 3, n = 5;
    for (int i = n; i > pos; i--) {
        arr[i] = arr[i-1];
    }
    arr[pos] = num;
    n++;
    for (int i = 0; i < n; i++) {
        printf("%d", arr[i]);
    }
    return 0;
}
```

// 1 2 3 4 5 6

=> explain:
- we declare an array (arr) of ~~size~~ 6.
- we initialize only 5 elements, {1,2,4,5,6} meaning 6th element is uninitialized
- total no. of elements currently in use is (n=5);
- we want to insert 3, at ~~position~~ 2.
- n=5 represents current no. index of elements in use

+91-8233266276
info@grootacademy.com
122/166, 2nd Floor, vijay path, Mansarovar, 302020
www.Grootacademy.com

⊙ Lifting elements & making space

↳

index	bef. sh.	after sh.
0	1	1
1	2	2
2	4	4→5
3	5	5→6
4	6	6
5	unused	6

↳ arr = {1, 2, 4, 4, 5, 6};

⊙ placing new element, arr[2] = 3, replacing old 4.

⊙ n++ → a new element added so incrementing the updated count.

⊙ displaying the updated array.

© deletion from an array

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int pos = 2, n = 5;        // delete element at index 2.
for (int i = pos; i < n-1; i++) {
        arr[i] = arr[i+1];
    }
n--;
for (int i=0; i<n; i++) {
    printf("%d", arr[i]);
}
return 0;                      //10 20 40 50
}
```

GROOT ACADEMY

Searching in Arrays

a) Linear search

```
#include <stdio.h>
int linearSearch(int arr[], int n, int key) {
    for (int i=0; i<n; i++) {
        if (arr[i] == key) return i;
    }
    return -1;
}

int main() {
    int arr[] = {1,2,3,4,5};
    int key = 30;
    int index = linearSearch(arr, 5, key);
    if (index != -1)
        printf("element found at index : %d", index);
    else
        printf("element not found");
    return 0;
}
```

// found at index 2

GROOT ACADEMY

Strings in C

a) Reading a string from the terminal :

```
#include <stdio.h>
int main () {
    char name[20];
    printf ("enter your name : ");
    scanf ("%s", name);      // reads a single word (stops at space)
    printf ("Hello, %s!\n", name);
    return 0;
}
```

issue - with "lokesh yadav" → read until lokesh only

fix - use " fgets() to read full line.

↳ instead of scanf → fgets

```
fgets (name, sizeof(name), stdin);
printf ("Hello, %s", name);
return 0;
}
```

b) Writing the string to the screen :

↳ using printf ();

```
#include <stdio.h>
int main () {
    char str[] = "C programming";
    printf ("the string is : %s\n", str);
    return 0;
}
```

c) String Handling functions: #include

↳ C provides useful library <string.h> for useful functions:

↳ strlen() — find length

```
#include <stdio.h>
#include <string.h>
int main () {
    char str[] = "Hello";
    printf ("length : %d \n", strlen (str));
                                    //output : 5
    return 0;
}
```

↳ strcpy () — copy string

```
#include <stdio.h>
#include <string.h>
int main () {
    char src[] = "Laksh";
    char dest [20];
    strcpy (dest, src);          // copy src → dest
    printf ("copied string : %s \n", dest);
    return 0;
}
```

↳ strcat () — concatenate (join) strings

```
#include <stdio.h>
#include <string.h>
```

```
int main () {
    char str1 [20] = "Hello";
    char str2 [ ] = "Lakki";

strcat (str1, str2);    // appends str2 → str1
printf ("concated : %s \n", str1);
    return 0;
}
```

↳ strcmp() — compare strings

```
#include <stdio.h>
#include <string.h>
int main () {
        char1 [ ] = "appli";
        char2 [ ] =

    char str1 [=] = "apple";
    char str2 [ ] = "banana";
int result = strcmp (str1, str2);
if (result == 0)
        printf ("strings are equal");
    else if (result < 0)
            printf ("str1 comes before str2");

    else
        printf ("str1 comes after str2);

    return 0;
}
```

↳ finding a substring (strstr)

⇒ finds the first occurance of a substring.

⇒
```
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Program in C";
    char *sub = strstr(str, "in");
    if (sub)
        printf("substring found at : %s \n", sub);
    else
        printf("substring not found \n");
    return 0;
}
```

⊙ Table of string Functions

[function]	[Description]
1.> strcpy (dest, src)	Copies (src) string to (dest)
2.> strcat (dest, src)	Appends (src) string to (dest)
3.> strlen (str)	Returns the length of (str)
4.> strcmp (str1, str2)	Compares (str1) and (str2)
5.> strncmp (str1, str2, n)	Compares first (n) chara. of (str1) & (str2)
6.> strchr (str, ch)	finds 1st occurance of (ch) in (str)
7.> strrchr (str, ch)	finds last occurance of (ch) in (str)
8.> strstr (str, sub)	finds first occurance of (sub) in (str)