Introduction to Arrays

Definition of an array

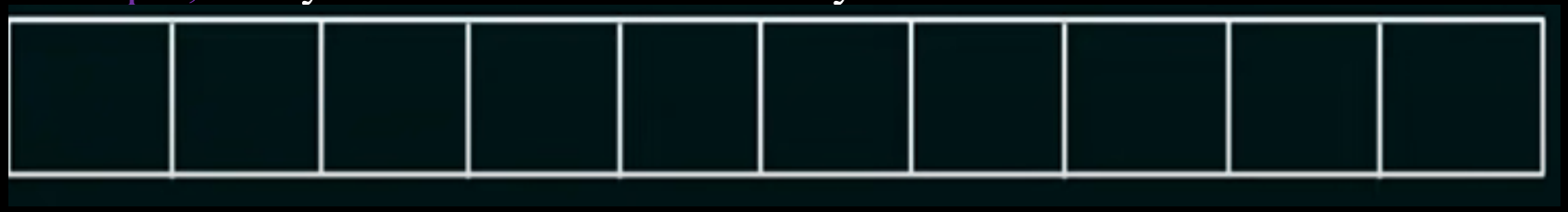
- An array is data structure containing a number of data values (all of which are of same type).

What is data structure

Data structure is a format for organize and storing data.

Also, each data structure is designed to organize data to suit a specific purpose.

for example, Array is a data structure which you can visualize as follow :



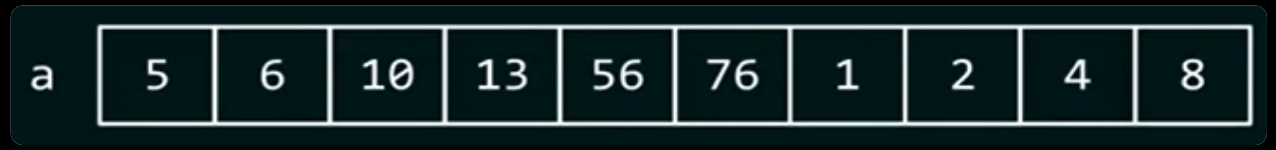
- Imagine an array is large chunk of memory dividing into small block of memory and each block can store value of some type.

Containing a number of data values

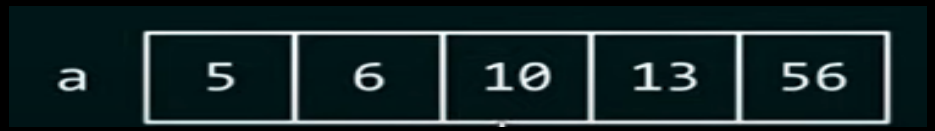
- An array can store **multiple data values.**
- Each value is stored in a **separate block of memory.**
- Hence, an array is said to contain a number of data values.

Example:

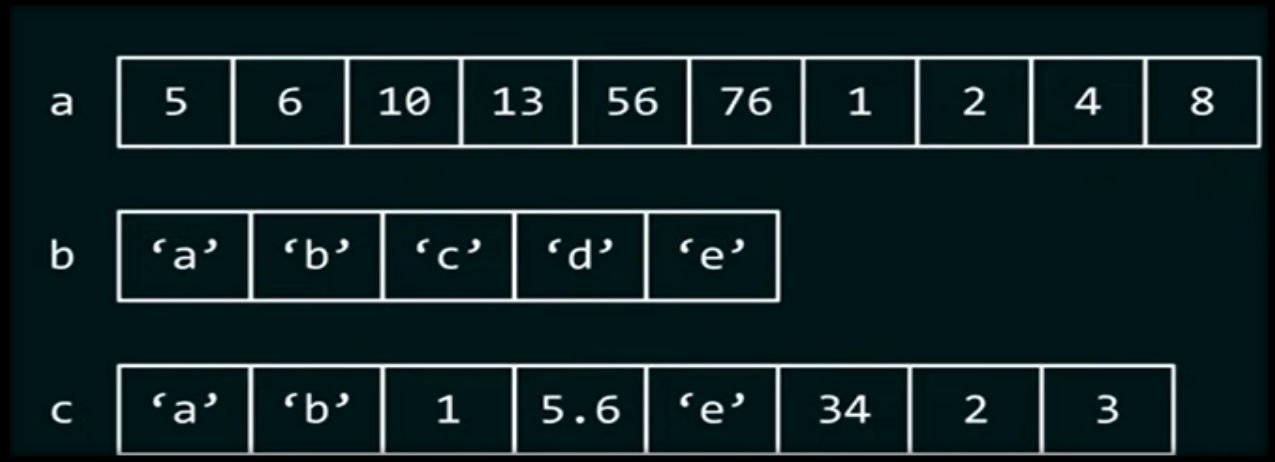
- This array consists of **10 data values**.



- Similarly, this array consists of **5 data values**.



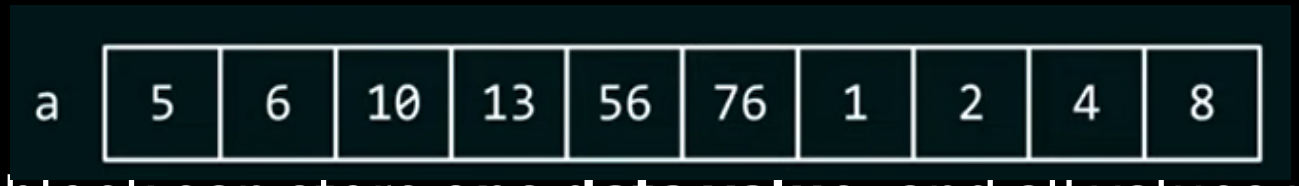
Data type of Array Elements



- **Array a =>** Correct, because all elements are of the same type (**integers**).
- **Array b =>** Correct, because all elements are of the same type (**characters**).
- **Array b =>** Incorrect, because it contains elements of **different data types** (characters, integers, float).
- In C, an array must always be **homogeneous**, meaning it can only store values of **one data type**.

One dimensional array

- **The simplest form of array one can imagine is one dimensional array.**
- A 1D array can be visualized as a **single row** of memory divided into blocks.



- Each block can store **one data value**, and all values must be of the **same type**.
- You can think of each block as a **variable**, and the array as a **collection of variables** stored together.

Declaration and definition of 1 D Array

- **Syntax: data_type name of the array[no.of elements];**

For example: an array of integer can be declared as follows:

- When we declaring the array, we defining the array also.
- The compiler allocates **contiguous memory** of size = 5*sizeof(int).
- If sizeof(int)=4,then the total memory = 5*4=20 bytes.
- Note sizeof depends on the machine (may vary).

Declaration and definition of 1 D Array

The length of an array must be positive integer constant.

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

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

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

```
Int a;
int arr[a=10/2];
```

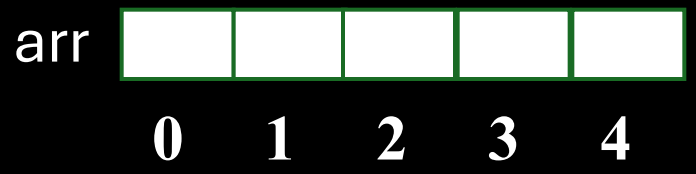
- Negative size is **not allowed.**

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

Accessing the element from 1 D Array

To access an array elements , just write.

array_name[index]



Accessing the first element of an array arr[0]
Accessing the second element of an array arr[1]

- **Note array index start from 0 and goes up to length-1**

Initializing 1 D Array

- There are 4 methods to initializing the 1 d array.
- **Method 1:**
- **int arr[5] = {1, 2, 5, 67, 32};**
- Specify data type, name, size, and elements inside { }
- Elements are **comma separated** and stored from index 0 onwards
- maximum size = 5, values stored at index 0 → 4.

- Method 2:

int arr[] = {1, 2, 5, 67, 32};

- No need to specify size.
- Compiler automatically assigns size = number of elements
- Advantage: more flexible.

Initializing 1 D Array

Method 3:

```
int arr[5];
arr[0] = 1;
arr[1] = 2;
arr[2] = 5;
arr[3] = 67;
arr[4] = 32;
```

- Declare first, then assign values one by one.
- Assign values one by one using index

Method 4:

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

- Declare the array, then take **user input** using a loop
- Use for loop with scanf to take values from user.
- Each value is stored in next index

Special Cases in Array Initialization

Q what if number of element are lesser than the length specified?

A If elements < size → remaining positions are filled with **0**.

```
int arr[10] = {45, 6, 2, 78, 5, 6};
// becomes {45, 6, 2, 78, 5, 6, 0, 0, 0, 0}
```

1. Cannot initialize with completely empty braces:

```
int arr[10] = {};   // Invalid
```

This is **illegal** because at least **one element** must be specified.

2. Cannot give **more elements** than size:

```
int arr[5] = {1, 2, 3, 4, 5, 6};   // Error
```

Multi dimensional array

Multidimensional arrays can be defined as array of arrays .

General form of declaring N-dimensional array is follows:

```
data_type  name_of_array[size1][size2].....[sizeN];
```

for example,

int a[3][4];//Two-Dimensional Array

int a[3][4][6]//Three-dimensional Array

Size of **Multi dimensional array**

Size of multidimensional array can be calculated by multiplying the size of all the dimensions.

- **Number of elements** = product of all dimensions
- **Size in bytes** = (number of elements) × (size of data type)

for example,

1. int arr[10][20];

Elements = 10 × 20 = **200**

Size = 200 × 4 = **800 bytes** (if int =4 bytes)

2. int arr[4][10][20];

Elements = 4 × 10 × 20 = **800**

Size = 800 × 4 = **3200 bytes**

Size of **Multi dimensional array**

Calculating Size:

- Step 1: Multiply all dimensions to get the **total number of elements**.
- Step 2: Multiply by the **size of data type** to get **total bytes**.
- Example : a[10][20] -> 10×20=200 elements. Each int = 4 bytes → 200×4=800 bytes.

Two dimensional array

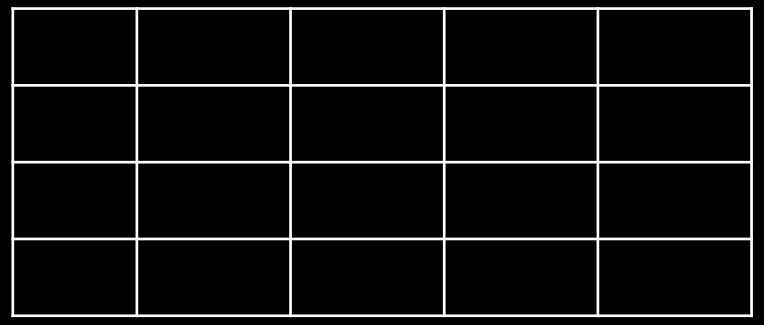
Syntax

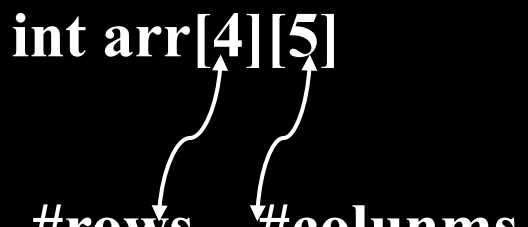
data_type name_of_array[x][y]

- **first, we must specify the data type and name then two square brackets.**
- **Where x and y are representing the size of the array**

Visualizing Two Dimensional Array

Recall that a multidimensional array is an array of array





Size of array [4][5] => 4*5 = 20 elements.

Initialize two dimensional Array

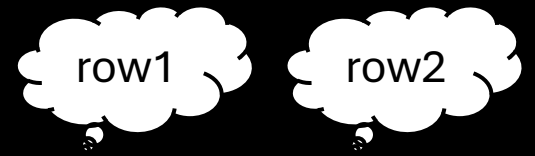
Method 1

int arr[2][3]={1,2,3,4,5,6};

	0	1	2
0	1	2	3
1	4	5	6

- Elements are stored **row by row** in consecutive memory locations.
- Harder to visualize

Method 2:



Int arr[2][3={{1,2,3},{4,5,6}};

	0	1	2
0	1	2	3
1	4	5	6

Same initialization as Method 1.

Easier to **visualize row-wise.**

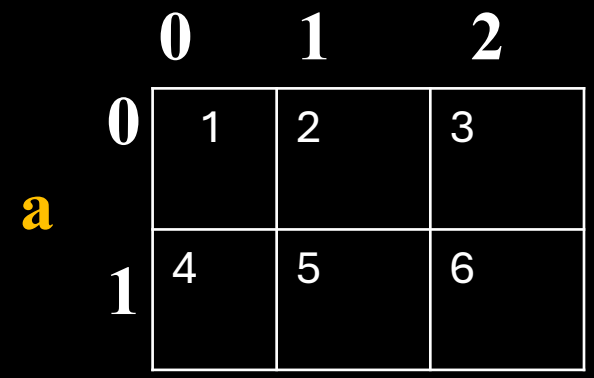
More readable and matches the row–column structure.

Access 2D Array Elements

Using row index and column index

For example:

We can access element stored in Ist row and 2nd column of below array



a [0][1]

- A[0][1] => elements at 1st row ,2nd column

Print two dimensional Array

1D array elements can be printed using single for loop.

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

    for(i=0; i<5;   i++){


}
```

2d array elements can be printed using two nested for loops.

```
int a[2][3]={{1 , 2 , 3},
                    {4 , 5 , 6}};

for( i=0 ; i<2  ; i++) {

for(j=0 ; j<3 ; j++) {

printf("%d", a[i][j]);
}
}
```

Dry run to print two dimensional Array elements

```
for( i=0 ; i<2  ; i++) {

for(j=0 ; j<3 ; j++) {

printf("%d", a[i][j]);
}
}
```

	0	1	2
0	1	2	3
1	4	5	6

Output : 1 2 3 4 5 6

i=0 j=0 a[0][0]

i=0 j=1 a[0][1]

i=0 j=2 a[0][2]

i=1 j=0 a[1][0]

i=1 j=1 a[1][1]

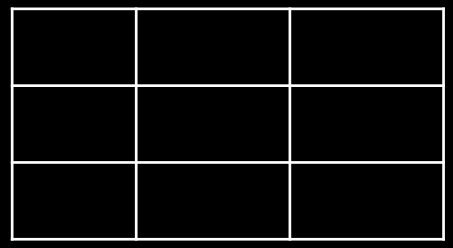
i=1 j=2 a[1][2]

Visualizing Three Dimensional Array

#rows

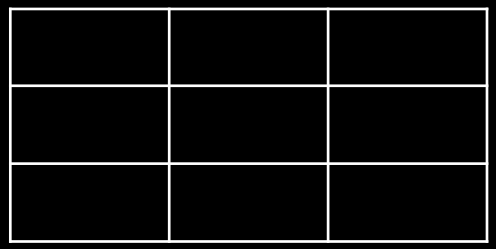
int arr[3][3] #columns

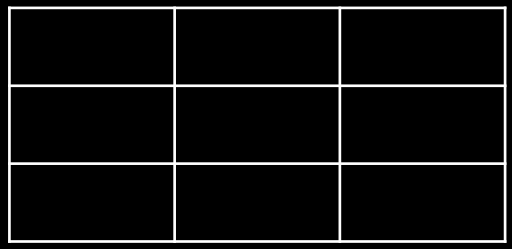
Recall that a 2 D Array consists of rows and columns



- 3x3
- A **2D array** → rows × columns (matrix).

int arr[2][3][3]





3X3

3X3

A **3D** array → collection of 2D arrays.

Means **2 blocks** of 2D arrays.

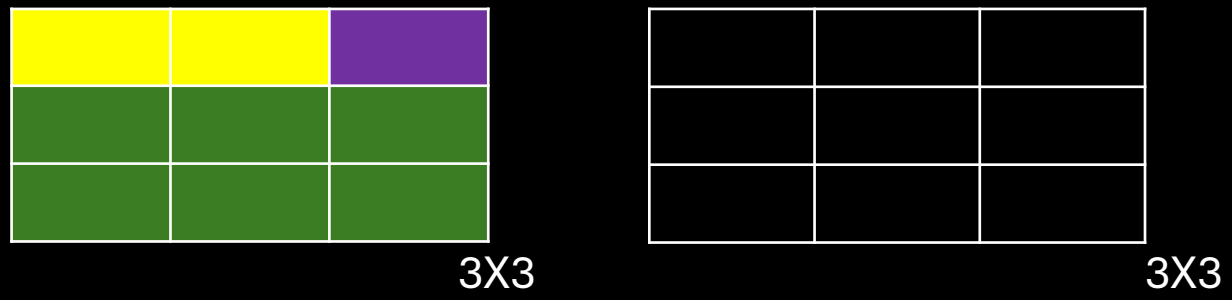
Each block = 3 rows × 3 columns.

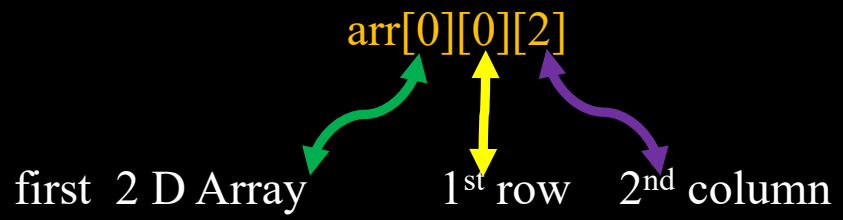
Total elements = 2 × 3 × 3 = **18**.

Accessing the 3 D Array elements

Suppose we want to access the element in the 1st row and 3rd column of 1st 2d array

int arr[2][3][3]





- 0 → 1st 2D array (block 1)
- 0 → 1st row
- 2 → 3rd column

Refers to the element at **1st row, 3rd column of 1st matrix**

Initialize 3 D Array elements

Method 1: (Sequential):

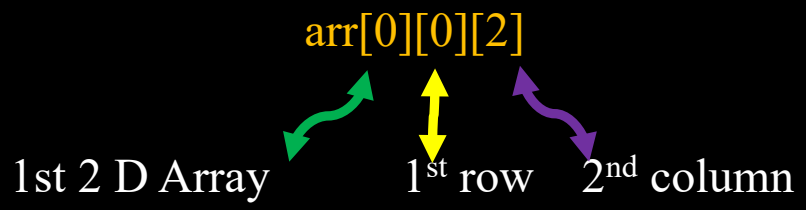
int a[2][2][3] = {1,2,3,4,5,6,7,8,9,10,11,12};

1	2	3
4	5	6

2X3

7	8	9
10	11	12

2X3



- Elements stored **contiguously** in memory
- Compiler fills elements **sequentially** (row by row, matrix by matrix).
- Difficult to **visualize row-wise**

Initialize 3 D Array elements

Method 2: (Sequential):

```
int a[2][2][3] = {
                    { {1,2,3}, {4,5,6} },
                    { {7,8,9}, {10,11,12} }
                  };
```

1	2	3
4	5	6

2X3

7	8	9
10	11	12

2X3

- Easier to **visualize as matrices**.
- Each inner block { } → one **row of a 2D array**.
- Outer blocks → represent **multiple matrices**.

Initialize 3 D Array elements

Use **three nested loops** (for blocks, rows, and columns).

```
int a[2][2][3] = {
  {{1, 2, 3}, {4, 5, 6}},
  {{7, 8, 9}, {10, 11, 12}}
};

for(i = 0; i < 2; i++) {        // block (2D arrays)
  for(j = 0; j < 2; j++) {      // rows
    for(k = 0; k < 3; k++) {    // columns
      printf("%d ", a[i][j][k]);
    }
    printf("\n");   // new row
  }
  printf("\n");     // new 2D array
}
```

1. In this example:

Array[2][2][3]

- 2 blocks (two 2D arrays).
- Each block → 2 rows × 3 columns.

2. To print a **3D array**, we need **three indexes** → block, row, column.

- I → which 2D array (block).
- J → row of that 2D array.
- K → column of that row.

3. Printing process:

- Outer loop (i) → selects the block.
- Middle loop (j) → selects the row.
- Inner loop (k) → prints each column element.

4. Extra printf(“\n”) ; ensures elements are printed in **matrix form**.

Operations on Array

Arrays allow us to perform various operations for data manipulation and access.
Here are **4 common operations** performed on 1D arrays:

1. Traversing (Accessing Elements)

- Traversing means **visiting each element** of the array one by one.
- Used to **display** or **process** all elements.
- Generally done using a **loop** (for or while).

Example:

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

Operations on Array

2. Insertion (Adding an Element)

- Used to **insert a new element** at a specific position in the array.
- Elements after that position are **shifted right** by one place.

Example:

```
for(i = n; i > pos; i--) {
    arr[i] = arr[i-1];
}
arr[pos] = value;
n++;
```

Operations on Array

3. Deletion (Removing an Element)

- Used to **remove an element** from a specific
- Elements after that position are **shifted left** to fill the gap.
- **Example:**

```
for(i = pos; i < n - 1; i++) {
    arr[i] = arr[i + 1];
}
n--;
```

Operations on Array

4. Searching (Finding an Element)

- Used to **locate the position** of a particular element in the array.
- Two main types:
  - **Linear Search** (sequential)
  - **Binary Search** (for sorted arrays)

- **Example(Linear Search):**

```
for(i = 0; i < n; i++) {
    if(arr[i] == key)
        printf("Found at %d", i);
}
```

What we have learned ?

1. How to declare and define 1,2 ,3-d array.
2. How to visualize 1,2 ,3-d array.
3. How to initialize 1,2 ,3-d array.
4. How to access 1,2 ,3-d array elements.
5. How to print 1,2 ,3-d array elements.