🧠 DSA Notes – Arrays (C++)

📌 What is DSA?

- **DSA** stands for **Data Structures and Algorithms**.
- **Data Structures** are ways to store and organize data.
- **Algorithms** are step-by-step methods to perform operations on data (like searching or sorting).

📦 Why Data Structures?

- Real-life systems (apps, websites, software) depend on **data**.
- Efficiently storing and handling large data is essential for performance and simplicity.

🗂 Arrays – The First Data Structure

✅ Definition:

An **Array** is a collection of **similar type of elements**, stored in **contiguous memory locations**, accessed using an **index**.

✅ Why Arrays?

Without arrays, we would need to create separate variables for each data item (e.g., `marks1`, `marks2`, ..., `marks100`), which is inefficient.

✅ Syntax:

```cpp
int marks[5]; // Declares an array of size 5 of type int
```

✅ Initialization:

```cpp
int marks[5] = {99, 100, 54, 36, 88};
double price[] = {98.9, 105.6, 30.00}; // Size auto-detected as 3
```

🔢 Array Properties

Property	Description
Same data type	All elements must be of the same type.
Contiguous memory	All elements are stored next to each other in memory.
Linear structure	Elements are stored in a linear (sequential) order.

🧮 Memory Example:

- Each `int` takes 4 bytes.
- If an array starts at address `100`:
  - 1st element → `100`
  - 2nd element → `104`
  - 3rd element → `108`, etc.

🧾 Accessing Array Elements

- Array indexing starts from **0**.

```
cout << marks[0]; // Prints first element
marks[0] = 101;   // Changes value of first element
```

- Invalid index (e.g., `marks[5]`) leads to **error or garbage value**.

🔁 Traversing Arrays Using Loops

```
for(int i = 0; i < 5; i++) {
    cout << marks[i] << endl;
}
```

📥 Taking Input in Array

```
int marks[5];
for(int i = 0; i < 5; i++) {
    cin >> marks[i];
}
```

📐 Finding Size of an Array

```
int size = sizeof(marks) / sizeof(marks[0]);
```