Session 6: Arrays in C

1. Title + Big Idea (The Setup)

Topic Name: Arrays in C

One-line purpose: To understand how to store and manage collections of similar data types efficiently using arrays in C programming.

2. Why Do We Need It? (The Problem)

Problem it solves: When dealing with a large number of related variables (e.g., scores of 100 students, names of 50 employees), declaring individual variables for each becomes impractical and cumbersome. Arrays provide a single, organized structure to hold such collections.

Real-life analogy: An array is like a row of mailboxes in an apartment building. Each mailbox has a unique number (index) and can hold the same type of item (mail). Instead of having separate mailboxes scattered everywhere, they are organized in a contiguous block.

3. Core Concept / Definition (The Truth)

Simple definition: An **array** is a collection of elements of the same data type, stored at contiguous memory locations. Each element can be accessed using an index, which typically starts from 0.

Key rule(s):

- All elements in an array must be of the **same data type**.
- Array elements are accessed using a **zero-based index** (the first element is at index 0).

- The **size of an array** must be a constant value known at compile time (for static arrays).
- Arrays are passed to functions **by reference** (the address of the first element is passed), allowing modifications within the function to affect the original array.

4. Visual / Flow / Diagram (Show, Don't Tell)

Array Representation:

```
Array Name: `scores`
Size: 5
Data Type: int

Memory Address:  1000   1004   1008   1012   1016
                +------+------+------+------+------+
Elements:       |  85  |  92  |  78  |  95  |  88  |
                +------+------+------+------+------+
Indices:          [0]    [1]    [2]    [3]    [4]
```

5. Syntax / Structure (The Tool)

Declaration:

```
data_type array_name[size];
// Example:
int numbers[10]; // Declares an integer array named 'numbers' of size 10
char name[50];   // Declares a character array (string) named 'name' of size
50
```

Initialization:

- **At declaration:**

```c
int scores[5] = {85, 92, 78, 95, 88};
char vowels[] = {'a', 'e', 'i', 'o', 'u'}; // Size is automatically
determined
```

- **After declaration (element by element):**

```c
int arr[3];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
```

Accessing Elements:

```c
array_name[index];
// Example:
int first_score = scores[0]; // Accesses the first element (85)
scores[2] = 80;              // Modifies the third element to 80
```

6. Example (The Action)

```c
#include <stdio.h>

int main() {
    // Declare and initialize an array of integers
    int student_ages[5] = {18, 19, 20, 18, 21};
    int i;

    printf("Student Ages:\n");
    // Loop through the array to print each element
    for (i = 0; i < 5; i++) {
        printf("Age of student %d: %d\n", i + 1, student_ages[i]);
    }

    // Calculate the average age
    int sum_ages = 0;
    for (i = 0; i < 5; i++) {
        sum_ages += student_ages[i];
    }
    float average_age = (float)sum_ages / 5;
    printf("\nAverage age of students: %.2f\n", average_age);

    // Modify an element
    student_ages[2] = 22; // Student 3's age was updated
    printf("Updated age of student 3: %d\n", student_ages[2]);

    return 0;
}
```

7. Common Mistakes (The Conflict)

1. **Array Index Out of Bounds:** Attempting to access an element using an index that is outside the valid range (0 to `size-1`). This leads to undefined behavior, crashes, or corrupted data.
2. **Fixed Size Limitation:** Forgetting that static arrays have a fixed size determined at compile time. You cannot resize them during runtime. For dynamic sizing, pointers and dynamic memory allocation (`malloc`, `calloc`) are needed.

3. **Passing Arrays to Functions:** Misunderstanding that when an array is passed to a function, it decays into a pointer to its first element. The function does not know the original size of the array, so the size often needs to be passed as a separate argument.
4. **Not Initializing Arrays:** If not explicitly initialized, local arrays contain garbage values, which can lead to unexpected program behavior.

8. Key Points / Rules (The Takeaway)

- Arrays store multiple values of the same data type in contiguous memory.
- Elements are accessed via zero-based indices.
- Declare arrays with a fixed size or let the compiler determine it during initialization.
- Be extremely careful to avoid array index out-of-bounds errors.
- When passing arrays to functions, remember that they are passed as pointers, and their size needs to be managed explicitly.

9. One-Line Summary (The Ending)

Arrays provide a fundamental and efficient way to organize and manipulate collections of homogeneous data, making it easier to handle large sets of related information in C programs.

10. (Optional) Practice Trigger

Question: Write a C program that takes 5 integer inputs from the user, stores them in an array, and then prints the largest and smallest numbers from that array.

Variation Idea: Extend the program to also calculate and print the sum and average of the numbers stored in the array.