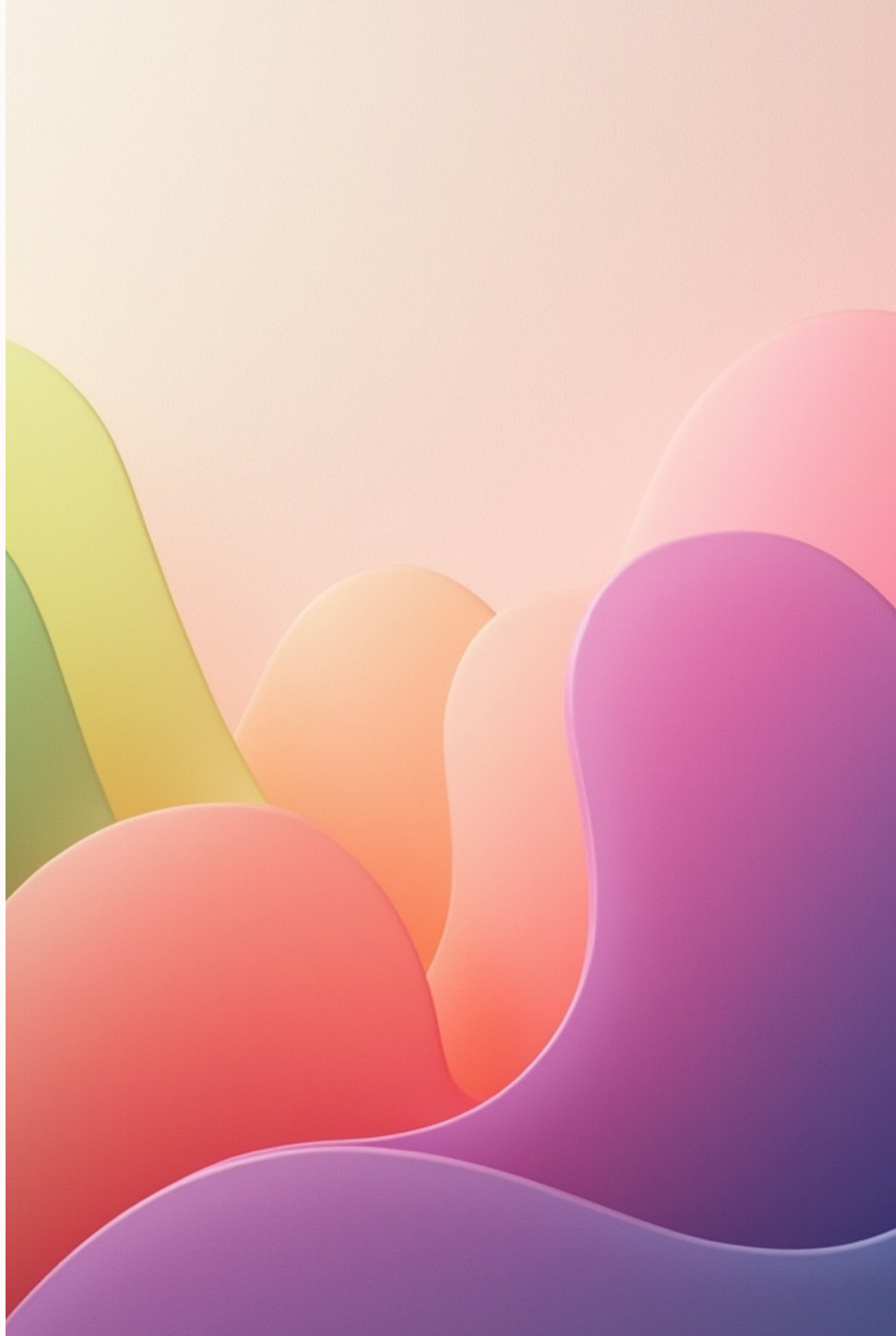
Session 6: Arrays in C

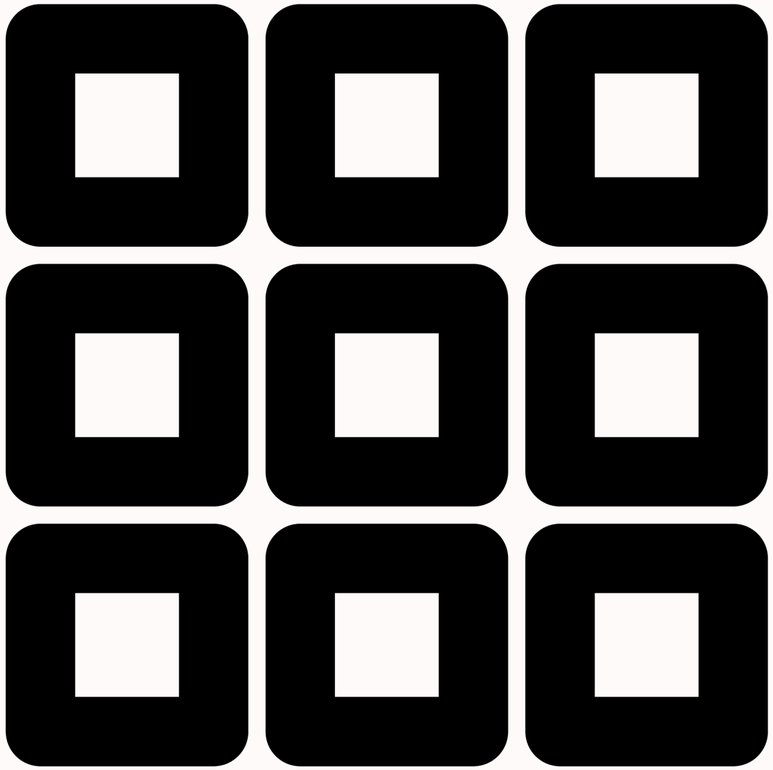
covering how to declare, initialize, and access fixed-size collections of data stored in contiguous memory.

TeachToTech



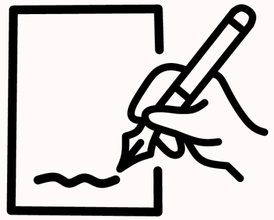
What is an Array?

An array is a collection of elements of the same data type, stored in contiguous memory locations.
Instead of many variables, one name holds many values.



Why Arrays are Important

- Store large data efficiently
- Enable searching and sorting
- Foundation for strings, matrices, and algorithms
- Reduce code complexity



Array Declaration

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

- int → data type
- arr → array name
- 5 → size (fixed)

Array Initialization.c

```
#include <iostream>

int main() {
    // Declare and initialize the array
    int arr[5] = {10, 20, 30, 40, 50};

    // Print the elements of the array
    std::cout << "Elements of the array are: " << std::endl;
    for (int i = 0; i < 5; ++i) {
        std::cout << "arr[" << i << "] = " << arr[i] <<
std::endl;
    }

    return 0;
}
```

Accessing Array Elements

Accessing Array.c

```
printf("%d", arr[0]);   // first element
printf("%d", arr[4]);   // last element
```

Traversing an Array

Traversing Array.c

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

Passing Array to Function

Passing Array to Function.c

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

2D Arrays (Matrices)

2D arrays.c

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

Traversing 2D Arrays

Traversing 2D Arrays.c

```
for(int i=0;i<2;i++){
    for(int j=0;j<3;j++){
        printf("%d ", mat[i][j]);
    }
}
```

Sequential (Linear) Search

Array Initialization.c

```
int key = 5;
for(int i=0;i<n;i++){
    if(arr[i] == key){
        printf("Found at %d", i);
        break;
    }
}
```

Binary Search (Iteration)

Array Initialization.c

```
int low=0, high=n-1, mid;
while(low <= high){
    mid = (low + high) / 2;
    if(arr[mid] == key)
        break;
    else if(arr[mid] < key)
        low = mid + 1;
    else
        high = mid - 1;
}
```

Binary Search (Recursion)

Array Initialization.c

```
int binarySearch(int arr[], int low, int high, int key){
    if(low > high)
        return -1;
    int mid = (low + high) / 2;
    if(arr[mid] == key)
        return mid;
    if(arr[mid] > key)
        return binarySearch(arr, low, mid-1, key);
    return binarySearch(arr, mid+1, high, key);
}
```

Array Declaration

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

- int → data type
- arr → array name
- 5 → size (fixed)

Array Initialization.c

```
#include <iostream>

int main() {
    // Declare and initialize the array
    int arr[5] = {10, 20, 30, 40, 50};

    // Print the elements of the array
    std::cout << "Elements of the array are: " <<
std::endl;
    for (int i = 0; i < 5; ++i) {
        std::cout << "arr[" << i << "] = " << arr[i] <<
std::endl;
    }

    return 0;
}
```

Array Problems from List

- Remove Element
- Remove Duplicates from Sorted Array
- Plus One
- Contains Duplicate
- Rotate Array
- Intersection of Two Arrays
- Search in Rotated Sorted Array

Activity:
Find Largest Element

Array Initialization.c

```
int max = arr[0];
for(int i=1;i<n;i++){
    if(arr[i] > max)
        max = arr[i];
}
```

Common Mistakes with Arrays

- Accessing out-of-bound index
- Forgetting size
- Confusing array with pointer
- Not sorting before binary search

Session Summary

- Arrays store multiple values
- 1D and 2D arrays
- Searching techniques
- Foundation for advanced DSA