Arrays in C

Arrays are a fundamental data structure in C that allows us to store multiple values of the same data type under a single variable name. They are particularly useful when dealing with large amounts of data, such as lists of numbers, strings, or other data types. Let's explore arrays in detail with clear explanations and examples, keeping in mind that these notes are aimed at students who are new to programming in C.

1. Declaration of Arrays

An array is declared by specifying the type of its elements, followed by the array's name and its size (i.e., the number of elements it can hold) in square brackets.

Syntax:

data_type array_name[size];

Example:

int numbers[5]; // Declares an array of integers named 'numbers' with 5 elements.

2. Accessing and Storing Values in Arrays

Arrays are accessed using indices, which start from 0. This means the first element is at index 0, the second at index 1, and so on.

Example:

```
#include <stdio.h>
int main() {
    int scores[3]; // Declaring an array of 3 integers

    // Storing values in the array
    scores[0] = 85; // Assign 85 to the first element
    scores[1] = 90; // Assign 90 to the second element
    scores[2] = 78; // Assign 78 to the third element

    // Accessing and printing values from the array
    printf("First score: %d\n", scores[0]); // Outputs 85
    printf("Second score: %d\n", scores[1]); // Outputs 90
```

```
    printf("Third score: %d\n", scores[2]); // Outputs 78

    return 0;
}
```

Explanation:
- We declared an array `scores` of size 3 to store integer values.
- We assigned values to the array using indices.
- We accessed and printed the values using their respective indices.

3. Operations on Arrays

Arrays allow various operations, such as initializing, updating, and looping through the elements. Let's look at some common operations.

Initializing an Array:
Arrays can be initialized at the time of declaration.

```
int numbers[5] = {10, 20, 30, 40, 50}; // All elements are initialized
```

If you don't provide all the values, uninitialized elements default to zero.

Example of Summing Elements of an Array:

```
#include <stdio.h>
int main() {
    int numbers[5] = {10, 20, 30, 40, 50}; // Initializing an array
    int sum = 0;

    // Looping through the array to calculate the sum
    for (int i = 0; i < 5; i++) {
        sum += numbers[i];
    }

    printf("Sum of array elements: %d\n", sum); // Outputs 150
```

```
    return 0;
}
```

Explanation:

- We initialized an array `numbers` with 5 elements.
- We used a `for` loop to traverse the array, adding each element to the variable `sum`.
- Finally, we printed the sum of the array elements.

4. Passing Arrays to Functions

Arrays can be passed to functions to perform operations on them. Only the address of the first element is passed, so changes made to the array inside the function affect the original array.

Example:

```
#include <stdio.h>
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int numbers[4] = {1, 2, 3, 4};
    printArray(numbers, 4); // Passing array to a function
    return 0;
}
```

Explanation:

- The `printArray` function takes an array and its size as arguments.
- We loop through the array and print each element.
- Note: The array's size is not inherently known to the function; hence, we need to pass it explicitly.

5. Two-Dimensional Arrays

A two-dimensional array is like a matrix or a table with rows and columns. It is declared by specifying two sizes.

Syntax:

data_type array_name[rows][columns];

Example:

```
#include <stdio.h>
int main() {
    int matrix[2][3] = {
                        {1, 2, 3},
                        {4, 5, 6}
                       }; // Declaring a 2x3 matrix

    // Accessing and printing 2D array elements
    for (int i = 0; i < 2; i++) { // Looping through rows
        for (int j = 0; j < 3; j++) { // Looping through columns
            printf("%d ", matrix[i][j]);
        }
        printf("\n"); // Newline after each row
    }
    return 0;
}
```

Explanation:

- We declared a 2x3 matrix (2 rows and 3 columns).
- We used nested loops to access each element by row and column indices.
- The matrix is printed in a table format.

6. Operations on 2D Arrays

We can perform various operations, like matrix addition, multiplication, etc., on 2D arrays. Here's an example of adding two matrices.

Example: Matrix Addition

```
#include <stdio.h>
int main() {
    int A[2][2] = { {1, 2}, {3, 4} };
    int B[2][2] = { {5, 6}, {7, 8} };
    int C[2][2]; // Resultant matrix

    // Adding two matrices
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            C[i][j] = A[i][j] + B[i][j]; // Element-wise addition
        }
    }

    // Printing the result
    printf("Sum of matrices:\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("%d ", C[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Explanation:

- We declared two 2x2 matrices `A` and `B`.
- We performed element-wise addition to store the results in matrix `C`.
- We printed the resulting matrix `C`.

7. Multidimensional Arrays

Arrays can have more than two dimensions, though 2D arrays are most commonly used.

Multidimensional arrays are declared similarly but with more indices.

Syntax for 3D Array:

data_type array_name[size1][size2][size3];

Example:

```
#include <stdio.h>
int main() {
    int arr[2][2][3] = {
        {
            {1, 2, 3},
            {4, 5, 6}
        },
        {
            {7, 8, 9},
            {10, 11, 12}
        }
    };
    // Accessing 3D array elements
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 3; k++) {
                printf("%d ", arr[i][j][k]);
            }
            printf("\n");
        }
        printf("\n");
    }

    return 0;
}
```

Explanation:

- This is a 3-dimensional array with 2 layers, each having 2 rows and 3 columns.
- We used three nested loops to access each element of the 3D array.

8. Applications of Arrays

Arrays are used in numerous real-life applications and programming scenarios:

- Storing Data: Arrays are used to store data like scores, temperatures, stock prices, etc.
- Sorting and Searching: Arrays are fundamental to algorithms like sorting (Bubble Sort, Quick Sort) and searching (Binary Search).
- Game Development: Grids in games (like a chessboard) are represented using 2D arrays.
- Data Manipulation: Arrays are used to perform operations on large sets of data, such as mathematical computations and simulations.

Example of Array Application in Calculating Average Score:

```
#include <stdio.h>
int main() {
    int scores[5] = {88, 92, 78, 85, 91};
    int sum = 0;
    float average;

    // Calculating sum
    for (int i = 0; i < 5; i++) {
        sum += scores[i];
    }

    // Calculating average
    average = (float)sum / 5;

    printf("Average score: %.2f\n", average); // Outputs 86.80

    return 0;
}
```

Explanation:

- We stored 5 scores in an array and calculated the sum using a loop.
- We then calculated the average and printed it, demonstrating a common use of arrays in handling collections of data.