Short Note for Arrays

1 What Are the Arrays?

In programming, an array is a data structure that stores a collection of elements of the same data type in a contiguous block of memory. Each element in an array is accessed by an index or position, which typically starts at 0 for the first element. Arrays are a fundamental concept in programming and are used to store and manipulate collections of data.

- **Homogeneous Data**: Arrays store elements of the same data type, such as integers, characters, or floating-point numbers.
- **Fixed Size**: Arrays have a fixed size, meaning the number of elements they can hold is determined at the time of declaration and cannot be changed dynamically.
- **Index-Based Access**: Elements in an array are accessed using indices (positions), starting from 0 for the first element. For instance, the first element in an array is accessed using index 0, the second with index 1, and so on.
- **Contiguous Memory**: Array elements are stored in contiguous memory locations, which means they are stored one after the other in memory, making it efficient for direct access.
- **Declaration**: Arrays are declared by specifying the data type of their elements and their size. For example, int numbers[5] declares an array of integers with a size of 5.
- **Initialization**: Arrays can be initialized at the time of declaration, where you provide initial values for the elements. For instance, int numbers[5] = {1, 2, 3, 4, 5}; initializes an integer array with specific values.
- **Loops and Iteration**: Arrays are often used in combination with loops to iterate over their elements, making it easy to perform operations on each element.
- **Random Access**: Arrays allow for fast random access to elements because you can directly calculate the memory location of an element based on its index.
- **Multidimensional Arrays**: Arrays can have multiple dimensions, like 2D arrays (matrices) or 3D arrays. These are used for storing data in grid-like structures.
- **Strings as Arrays**: In many programming languages, strings are represented as arrays of characters. Each character in a string is stored in an array element, allowing for manipulation of text.

2 One Dimensional Array

2.1 Declaring and Initializing an Array

In C, you can declare an array by specifying its data type and size. Here's an example of declaring and initializing an array of integers:

```
int numbers[5];
```

You can also initialize the array at the time of declaration:

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

2.2 Accessing Array Elements

Array elements are accessed using their index, which starts at 0 for the first element. For example, to access the first element of the numbers array:

```
int firstElement = numbers[0];
```

2.3 Iterating Over Elements

Arrays are often used in conjunction with loops to perform operations on multiple elements. Here's an example that prints all the elements of the numbers array:

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

3 Two-Dimensional Array

3.1 Declaring and Initializing an Array

To declare a two-dimensional array, you specify the data type of its elements and the number of rows and columns. For example:

```
int matrix[3][4];
```

You can initialize a two-dimensional array at the time of declaration by providing values in nested braces:

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

3.2 Accessing Array Elements

Accessing elements in a two-dimensional array involves specifying both row and column indices. Remember that array indices start from 0. For example:

```
int element = matrix[1][2];
```

3.3 Iterating Over Elements

ou can use nested loops to traverse all elements in a two-dimensional array. Typically, one loop iterates through rows, and another loop iterates through columns:

```
for (int i = 0; i < 3; i++) {
   for (int j = 0; j < 4; j++) {
      printf("%d ", matrix[i][j]);
   }
   printf("\n");
}
```

4 Strings (Character Arrays)

In C, strings are represented as arrays of characters. Each character in a string is stored in a separate element of the array.

4.1 Declaration

To declare a string array, you define an array of characters and initialize it with a string literal or provide its size to accommodate the characters you intend to store:

```
char greeting[20];
```

4.2 Initialization

You can initialize a string array with a string literal:

```
char greeting[] = "Hello, World!";
```

Or you can initialize it using individual characters:

```
char greeting[20] = {'H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!', '\0'};
```

Null character ('\0') marking the end of the string.

4.3 Standard Library Functions for String Manipulation

In C, the **string.h** header is used to provide a set of functions for working with strings, character arrays, and related operations. It's an essential header for string manipulation in C programs.

In order to use these functions, you typically include the **string.h** header at the beginning of your C program:

```
#include <string.h>
```

Here are some of the commonly used functions and macros provided by the string.h header:

- strlen(): Calculates the length of a null-terminated string.
- strcpy(): Copies one string to another.
- strcat(): Concatenates (appends) one string to the end of another.

- strcmp(): Compares two strings lexicographically.
- strncmp(): Compares the first n characters of two strings.
- strncpy(): Copies up to n characters from one string to another.
- strchr(): Searches for the first occurrence of a character in a string.
- strrchr(): Searches for the last occurrence of a character in a string.
- strstr(): Searches for a substring in a string.
- strtok(): Splits a string into tokens based on a delimiter.
- strpbrk(): Searches a string for any character in a set.
- strcspn(): Calculates the length of the initial segment of a string that contains no characters from a specified set.
- strspn(): Calculates the length of the initial segment of a string that contains only characters from a specified set.

Ex 01:

```
char greeting[] = "Hello, World!";
size_t length = strlen(greeting);      // length will be 13
```

Ex 02:

```
char source[] = "Copy me!";
char destination[20];
strcpy(destination, source);      // destination now holds "Copy me!"
```