1. Arrays in C

Definition

An array is a collection of variables of the same type stored in contiguous memory locations. It allows you to store multiple values in a single variable.

Declaring and Initializing Arrays

To declare an array, you can use the following syntax:

```
data_type array_name[array_size];
```

Example:

```
int arr[5]; // Declares an array of 5 integers
```

You can also initialize an array during declaration:

```
int arr[5] = {1, 2, 3, 4, 5}; // Initializes with values
```

Size of an Array

The size of an array in bytes can be determined using the `sizeof` operator:

```
int arr[5];
printf("Size of array: %lu\n", sizeof(arr)); // Prints size in bytes
```

To find the number of elements in the array, you can do:

```
int arr[5] = {1, 2, 3, 4, 5};
int size = sizeof(arr) / sizeof(arr[0]); // Size of array
printf("Number of elements: %d\n", size);
```

Multi-dimensional Arrays

C allows multi-dimensional arrays, commonly used for matrix representations. You can declare a two-dimensional array like this:

```
int matrix[3][4]; // 3 rows and 4 columns
```

You can also initialize it:

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

2. Strings in C

In C, strings are represented as arrays of characters, terminated by a null character (`'\0'`).

Declaring and Initializing Strings

You can declare a string like this:

```c
char str[20]; // Array of characters
```

You can initialize a string as follows:

```c
char str[] = "Hello"; // Automatically adds the null character
```

Different Ways of Taking String Input

You can take string input using:

1. **scanf**:

   ```c
   char str[100];
   scanf("%s", str); // Reads a string until whitespace
   ```

2. **fgets**:

   ```c
   char str[100];
   fgets(str, sizeof(str), stdin); // Reads a line of text including spaces
   ```

Different Ways to Print Strings

You can print strings using:

1. **printf**:

   ```c
   printf("%s\n", str); // Prints the string
   ```

2. **puts**:

   ```c
   puts(str); // Prints and adds a new line automatically
   ```

String Operations

C provides several standard library functions for string manipulation in `<string.h>`:

- **strlen**: Returns the length of a string (not including the null terminator).

  ```c
  int length = strlen(str);
  ```

- **strcat**: Concatenates two strings.

  ```c
  char str1[20] = "Hello, ";
  char str2[] = "World!";
  strcat(str1, str2); // str1 now contains "Hello, World!"
  ```

- **strcmp**: Compares two strings.

  ```c
  int result = strcmp(str1, str2); // Returns 0 if equal, >0 if str1 > str2, <0 if str1 < str2
  ```

- **strcpy**: Copies one string to another.

  ```c
  char str1[20];
  strcpy(str1, str2); // str1 now contains a copy of str2
  ```

3. Pointers in C

Definition

A pointer is a variable that stores the memory address of another variable. Pointers are often used for dynamic memory allocation, arrays, and strings.

Size of Pointer Variable

The size of a pointer variable is determined using `sizeof`, typically 4 bytes on a 32-bit system and 8 bytes on a 64-bit system:

```
int *ptr;
printf("Size of pointer: %lu\n", sizeof(ptr)); // Outputs size of pointer
```

Referencing and Dereferencing

- **Referencing**: Using the `&` operator to get the address of a variable.

  ```
  int num = 10;
  int *ptr = &num; // ptr now holds the address of num
  ```

- **Dereferencing**: Using the `*` operator to access the value at the address stored in the pointer.

  ```
  int value = *ptr; // Gets the value at the address ptr points to (10)
  *ptr = 20; // Changes the value of num to 20
  ```

Summary

1. **Arrays:**
   - Declaration, initialization, finding size, and multi-dimensional arrays.
2. **Strings:**
   - Declaration, input methods, output methods, and common string operations.
3. **Pointers:**
   - Definition, size, referencing, and dereferencing.