DSA

Arrays in Data Structure | Declaration, Initialization, Memory representation

Jenny's Lectures CS IT

Understanding Memory and Arrays in Programming

In programming, memory is essentially a long tape of bytes, with each byte containing 8 bits. This can be extended to both sides, making it open-ended. To understand the need for arrays, we need to examine how areas can be declared, initialized, and represented in memory.

Storing Values in Memory

To store a value in memory, we need to know how much space will be allocated for it. For example, the data type int typically takes up 4 bytes to store an integer. The number 5 would need to be converted to binary, which is 32 bits or 4 bytes. In traditional compilers, we generally take 2 or 4 bytes to be the data type for storing numbers. So, if we were storing an integer, it would take up 2-4 bytes in memory.

The memory manager would allocate some memory for storing a variable, and the value stored in memory would be represented in binary. For example, the value stored in a variable could be 5, which would be represented as 101 in binary.

Using Arrays

An array is a collection of more than one element of the same datatype. For example, an array of characters would be of the data type char, and an array of integers would be of the data type int. The number of elements in an array is determined by the size of the array.

To declare an array in programming, we use a specific syntax. In C language, for example, we would write:

int n;

to declare an integer variable. To declare an array, we would use:

int a[16];

This creates an array called "a" with 16 elements.

Initializing Arrays

Arrays can also be initialized with values. For example, we could initialize an array of integers with the values 1, 2, and 3 like this:

int a[3] = {1, 2, 3};