Representing Arrays in Memory

To represent an array in memory, we need to know how the elements of the array are stored. In a one-dimensional array, the elements are stored in a single row with multiple columns.

Each element of the array takes up space in memory, depending on its data type. For example, an array of integers would take up 2-4 bytes of memory per element.

Overall, understanding memory and arrays is crucial to programming, as they are fundamental building blocks of many programs and applications.

Arrays in Memory

In this video, we will discuss how data is stored in arrays in memory. All the elements in an array are stored in consecutive/continuous locations with the index starting at zero. The array can be statically initialized at compile time or dynamically initialized at runtime.

One important point to note is that arrays are fixed-size. The elements are stored in sequential/continuous locations with each element taking up the same amount of memory.

Accessing Array Elements

The index of the array starts at zero (although it can start at one in some cases). The size of the array is the number of elements it can hold (n), with the index ranging from 0 to n-1. To access an element, use the formula: base address + (i * size of data type).

The array follows the random access method, and accessing an element has a time complexity of O(1).

Dynamic Allocation

The drawback of arrays is that the size needs to be specified at compile time, which may not always be possible. We may not know how much space we need until runtime. If we allocate more space than needed, there will be unused memory. If we allocate less space than needed, we will run out of memory.

To dynamically allocate memory, we use functions such as 'malloc' and 'calloc'. The amount of memory allocated is not contiguous, and the location of the data may not be known.