Arrays and Their Representation

---

1. Single-Dimensional Arrays

A **single-dimensional array** is a linear collection of elements, all of the same type, stored in contiguous memory locations. Each element in the array can be accessed by an index.

For example, a single-dimensional array `A` of size `n`:

```
A = [A0, A1, A2, ..., An-1]
```

In this array, the index of the first element is 0, and the index of the last element is `n-1`.

Index Formula for 1-D Array

The address of the element at index `i` in a one-dimensional array `A` can be computed using the following formula:

```
Address(A[i]) = BaseAddress + i * size_of_element
```

Where:

- `BaseAddress` is the starting address of the array.
- `i` is the index of the element.
- `size_of_element` is the size of each array element in bytes.

Example

Consider an array `A` of integers with a base address of 1000 and each element taking 4 bytes. If you want to find the address of `A[3]`:

```
Address(A[3]) = 1000 + 3 * 4 = 1012
```

2. Multi-Dimensional Arrays

A **multi-dimensional array** is an array of arrays. It can be 2D (matrix-like), 3D, or more. In the case of a 2D array, it's often represented as rows and columns.

For example, a 2D array `A` of size `m x n` (m rows, n columns) can be visualized as:

```
A = [ [A0, A1, ..., An-1],
      [A1, A2, ..., An-1],
      ...
      [Am-1, Am-2, ..., Amn-1] ]
```

Index Formula for 2-D Array

For a 2D array `A` with dimensions `m x n` (m rows and n columns), the address of the element `A[i][j]` can be calculated using the following formula:

```
Address(A[i][j]) = BaseAddress + (i * n + j) * size_of_element
```

Where:

- `i` is the row index.
- `j` is the column index.
- `n` is the number of columns.
- `BaseAddress` is the starting address of the array.
- `size_of_element` is the size of the array element in bytes.

Example

Consider a 2D array `A` with 3 rows and 4 columns, where the base address is 1000, and each element takes 4 bytes. If you want to find the address of `A[2][3]`:

```
Address(A[2][3]) = 1000 + (2 * 4 + 3) * 4 = 1000 + 11 * 4 = 1044
```

3. Multi-Dimensional Arrays

For **multi-dimensional arrays** (such as 3D arrays), the same principle applies, but the index is extended for more dimensions. A 3D array `A` of dimensions `p x q x r` can be represented as:

```
A = [[[A000, A001, ..., A00r-1],
      [A010, A011, ..., A01r-1],
      ...,
      [A0p-1, A0p, ..., A0p-r-1]],

     [[A100, A101, ..., A10r-1],
      ...,
     [A1p-1, A1r-1]]]
```

Index Formula for 3-D Array

For a 3D array `A` with dimensions `p x q x r`, the address of an element `A[i][j][k]` is computed as:

```
Address(A[i][j][k]) = BaseAddress + ((i * q * r) + (j * r) + k) * size_of_element
```

Where:

- `i` is the row index.
- `j` is the column index.
- `k` is the depth index.
- `p`, `q`, `r` are the respective dimensions of the array.
- `BaseAddress` is the starting address of the array.
- `size_of_element` is the size of the array element in bytes.

4. Representation of Arrays: Row Major and Column Major Order

Row Major Order

In **row-major order**, the elements of each row are stored in contiguous memory locations. After one row is completely stored, the next row is stored.

For example, a 2D array:

```
A = [[A00, A01, A02],
     [A10, A11, A12],
     [A20, A21, A22]]
```

In **row-major order**, the elements would be stored as:

```
[A00, A01, A02, A10, A11, A12, A20, A21, A22]
```

Column Major Order

In **column-major order**, the elements of each column are stored in contiguous memory locations. After one column is completely stored, the next column is stored.

For example, the same 2D array in **column-major order** would be stored as:

```
[A00, A10, A20, A01, A11, A21, A02, A12, A22]
```

Row Major and Column Major Order Formulae

The index formulae for accessing elements in row-major and column-major order are as follows:

Row Major Order Formula for a 2D array `A[i][j]` with dimensions `m x n`:

```
Address(A[i][j]) = BaseAddress + (i * n + j) * size_of_element
```

Column Major Order Formula for a 2D array `A[i][j]` with dimensions `m x n`:

```
Address(A[i][j]) = BaseAddress + (j * m + i) * size_of_element
```

Example for 2D Array (Row and Column Major)

Consider a 2D array `A` of size `3 x 3`, and we want to find the address of `A[2][1]`. Assume the base address is 1000, and each element takes 4 bytes.

1. **Row Major Order**:

   ```
   Address(A[2][1]) = 1000 + (2 * 3 + 1) * 4 = 1000 + (7) * 4 = 1000 + 28 = 1028
   ```

2. **Column Major Order**:

   ```
   Address(A[2][1]) = 1000 + (1 * 3 + 2) * 4 = 1000 + (5) * 4 = 1000 + 20 = 1020
   ```
