# Arrays

# Arrays – Definition

- Finite ordered collection of homogenous elements
- Static allocation
- Complexity of operations?
  - Access – Sequential and Random
  - Insertion
  - Deletion
  - Searching

# 1D Array : Accessing array elements

Example

{1, 2, 3, 4, 5}

Code Segment

```
for(i = 0; i < n; i++)
    printf("%d  ", a[i]);
```

Complexity

O(n)

# 2D Array : Accessing array elements

## Example

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 |

## Code Segment

```
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
        printf("%d  ", a[i][j]);
```

## Complexity

$O(n^2)$

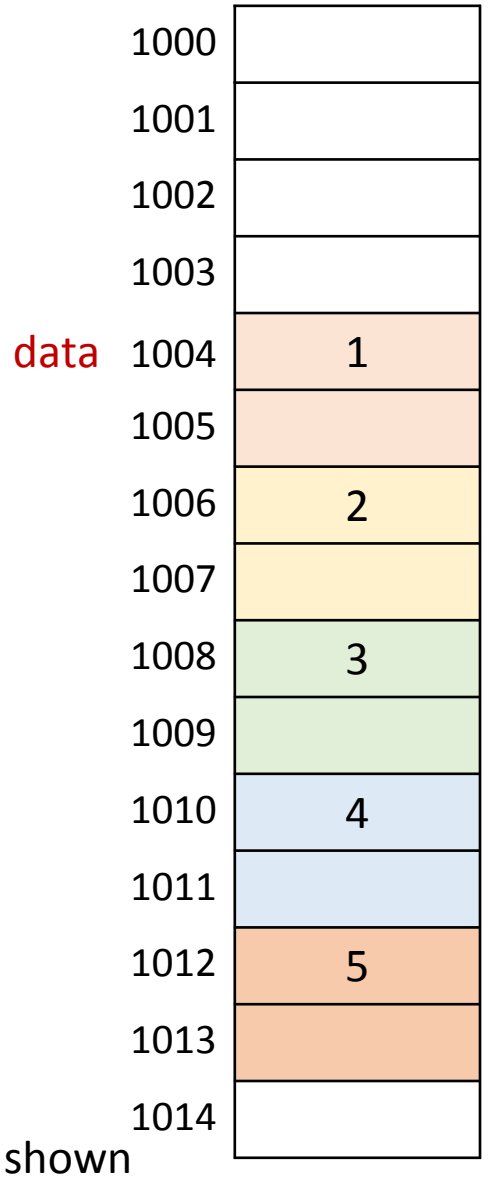
# Storage Representation : 1D array

Example

data = { 1, 2, 3, 4, 5}

Elements of arrays stored sequentially starting from the base address

| Element | Address |
|---|---|
| data[0] | 1004 |
| data[1] | 1006 |
| data[2] | 1008 |




Integer assumed to be 2 bytes; Little endian notation shown

# Storage Representation : 1D array

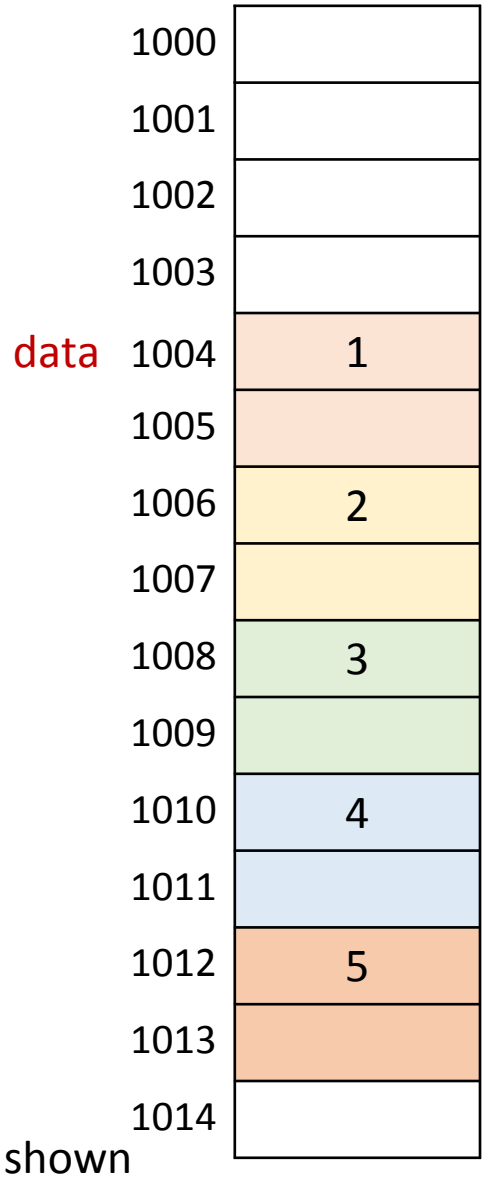
Example

data = { 1, 2, 3, 4, 5}

Elements of arrays stored sequentially starting from the base address

| Element | Address |
|---|---|
| data[0] | 1004 |
| data[1] | 1006 |
| data[2] | 1008 |

data[i] = base + i * element_size




Integer assumed to be 2 bytes; Little endian notation shown

# 2D Array

- Logical Representation – rows, columns
- Storage Representation – Single dimension memory

# Storage representation

| | |
|---|---|
| 100 | 10 |
| 102 | 20 |
| 104 | 30 |
| | 40 |
| | 50 |
| | 60 |
| | 70 |
| | 80 |

| | |
|---|---|
| 0 | 10 |
| 1 | 20 |
| 2 | 30 |
| 3 | 40 |
| 4 | 50 |
| 5 | 60 |
| 6 | 70 |
| 7 | 80 |

Base address = 100, size of integer is 2 bytes

Address of element at index 2 = 104 = 100 +2(2)

Addressing function

Address of $i^{th}$ element = base address + i * size of the element

# 2D Array – Storage representation

Representing 2 dimensional arrays in single dimension storage

- Row Major Order Representation
- Column Major Order Representation

# Row Major Order representation of 2D arrays

Matrix

| | | |
|---|---|---|
| 10 | 20 | 30 |
| 40 | 50 | 60 |
| 70 | 80 | 90 |

| | |
|---|---|
| 100 | 10 |
| 102 | 20 |
| 104 | 30 |
| 106 | 40 |
| 108 | 50 |
| 110 | 60 |
| 112 | 70 |
| 114 | 80 |
| 116 | 90 |

ROWS = 3, COLS = 3

Address of the element (0,2) = 104
Address of the element (1,2) = 110

Addressing function ?

# Column order representation of 2D arrays

Matrix

| | | |
|---|---|---|
| 10 | 20 | 30 |
| 40 | 50 | 60 |
| 70 | 80 | 90 |

| | |
|---|---|
| 100 | 10 |
| 102 | 40 |
| 104 | 70 |
| 106 | 20 |
| 108 | 50 |
| 110 | 80 |
| 112 | 30 |
| 114 | 60 |
| 116 | 90 |

ROWS = 3, COLS = 3

Address of the element (0,2) = 112
Address of the element (1,2) = 114

Addressing function ?

# Row Major Order representation of 2D arrays

Matrix

| | | |
|---|---|---|
| 10 | 20 | 30 |
| 40 | 50 | 60 |
| 70 | 80 | 90 |

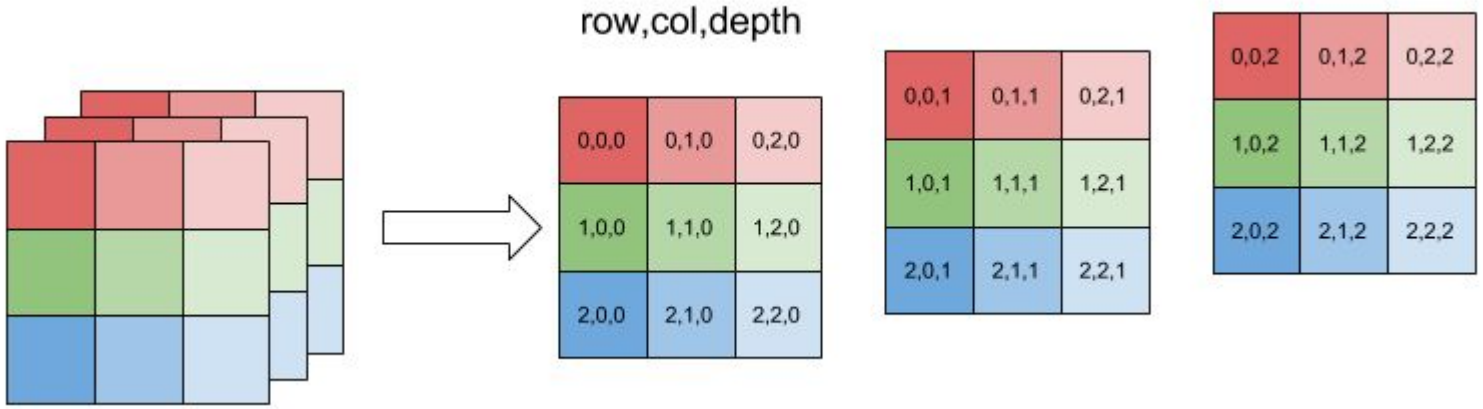
| | |
|---|---|
| 100 | 10 |
| 102 | 20 |
| 104 | 30 |
| 106 | 40 |
| 108 | 50 |
| 110 | 60 |
| 112 | 70 |
| 114 | 80 |
| 116 | 90 |

ROWS = 3, COLS = 3

Address of the element (0,2) = 104 = 100 + 2(2)
Address of the element (1,2) = 110 = 100 + 1 (3*2) + 2(2) = 100 + 2 ( 1* 3 +2 )

Addressing function = base address + size ( i * COLS + j )

# Storage representation of 3D arrays

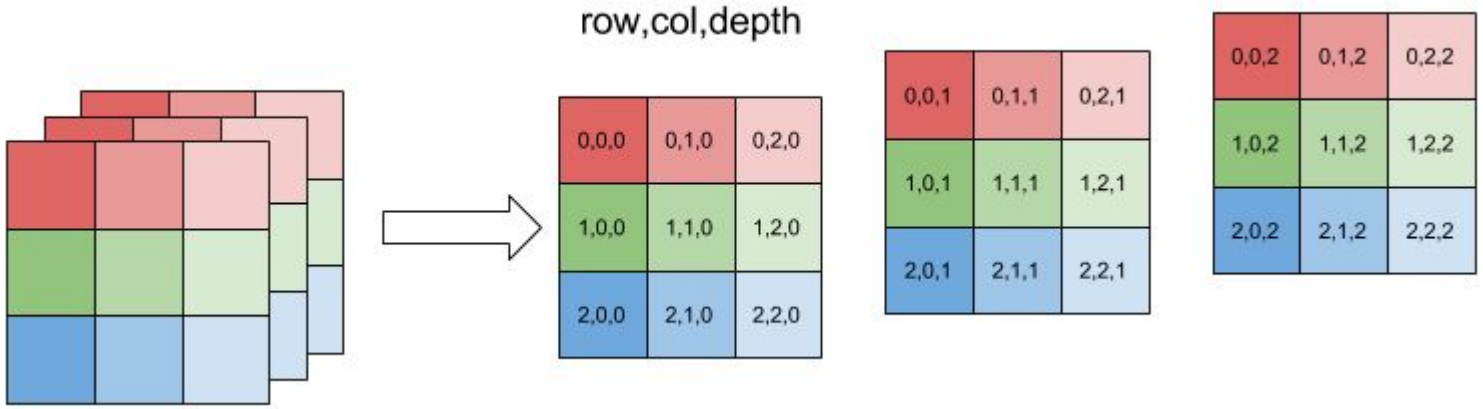



Store layer by layer – row/column major order

Row Major order

Address of (i, j, k) ?

# Storage representation of 3D arrays

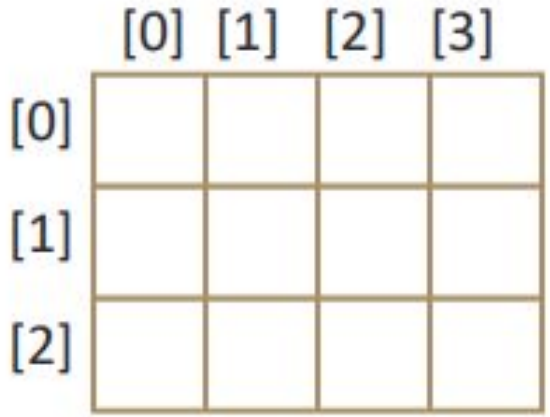


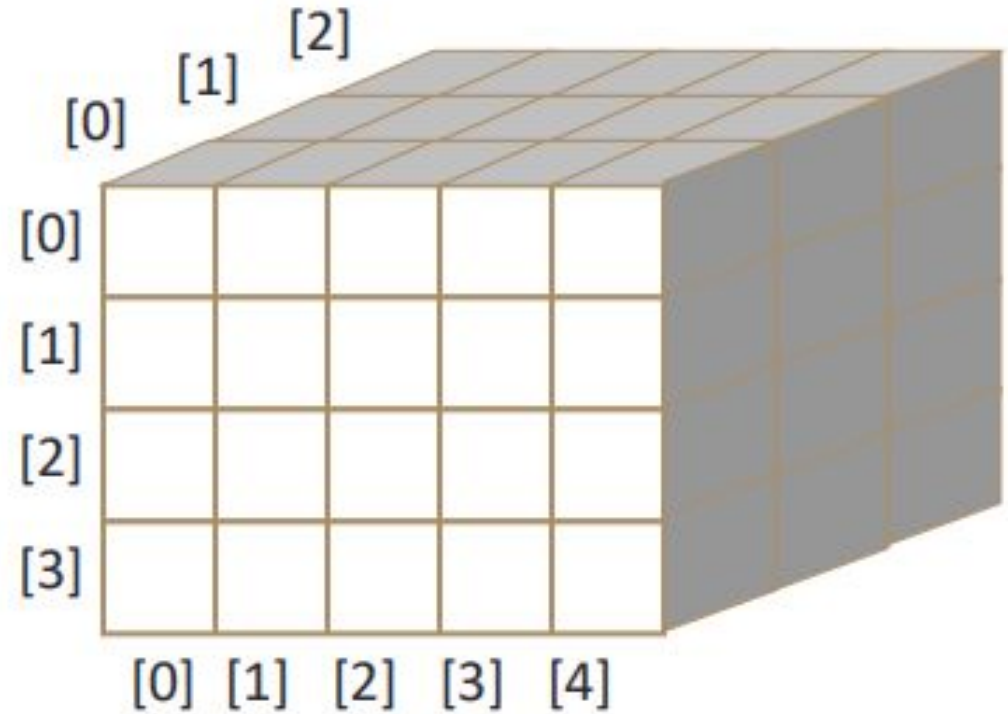

Store layer by layer – row/column major order

Row Major order

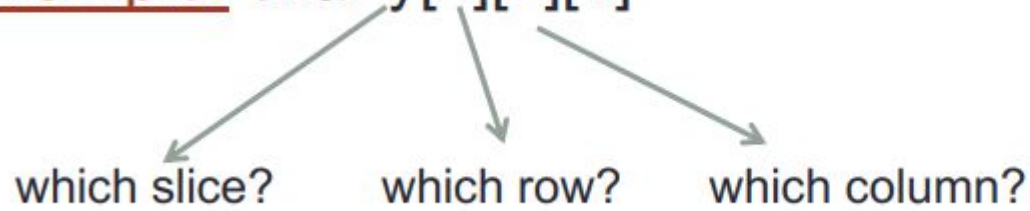
Address of (i, j, k) = base address + size ( k*ROWS * COLS + i*COLS + j )

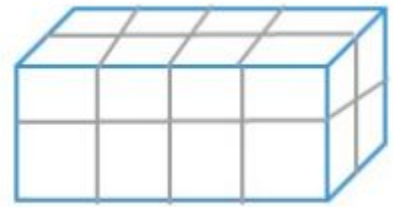
# Multi-dimensional Arrays



Example: char y[2][2][4]



# Memory Storage

The memory address of a[i][0][0] is:

$$\alpha + i * upper_1 * upper_2$$

if the memory address of a[0][0][0] is α. Therefore, the memory address of a[i][j][k] becomes:

$$\alpha + i * upper_1 * upper_2 + j * upper_2 + k$$

The memory address of $a[i_0]\,[i_1]\,[i_2]\ldots[i_{n-1}]$ is:

$$\alpha + \sum_{j=0}^{n-1} i_j a_j \begin{cases} a_j = \prod_{k=j+1}^{n-1} upper_k & 0 \le j \le n-1 \\ a_{n-1} = 1 \end{cases}$$

# Special matrices

Diagonal Matrix

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 0 | 2 | 0 | 0 |
| 0 | 0 | 3 | 0 |
| 0 | 0 | 0 | 4 |

Row Major Order

| |
|---|
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 2 |
| 0 |
| 0 |
| 0 |
| 0 |
| 3 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 4 |

Non zero elements : i = j

| |
|---|
| 1 |
| 2 |
| 3 |
| 4 |

Addressing Function

Address of (i,j) = base address + i * element size ; if i = j

A[i][j] = 0 ; otherwise

# Upper triangular matrices

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 0 | 5 | 6 | 7 |
| 0 | 0 | 8 | 9 |
| 0 | 0 | 0 | 10 |

4 x 4 matrix

| |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

# Upper triangular matrices

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 0 | 5 | 6 | 7 |
| 0 | 0 | 8 | 9 |
| 0 | 0 | 0 | 10 |

4 x 4 matrix

| |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

Condition for non zero element $i \leq j$

**Address of (2,3)**

No. of non-zero elements in rows 0 and 1 = ( 4 + 3 ) = 7

No. of preceding non-zero elements in row 2 = 1

Offset = 7 + 1 = 8

**Address of (3,3)**

No. of elements in rows 0,1 and 2 = ( 4 + 3 +2 ) = 9

No. of preceding non-zero elements in row 3 = 0

Offset = 9 + 0 = 9

# Addressing function of Upper triangular matrix

- Base address + element Size ( $\sum_{k=0}^{i} (ROW - k) + (j - i)$ )

# Lower triangular matrices

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 3 | 0 | 0 |
| 4 | 5 | 6 | 0 |
| 7 | 8 | 9 | 10 |

| |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

# Lower triangular matrices

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 3 | 0 | 0 |
| 4 | 5 | 6 | 0 |
| 7 | 8 | 9 | 10 |

| |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

Condition for non zero element $i \geq j$

**Address of (2,2)**

No. of elements in rows 0,1 = ( 1 + 2) = 3

No. of preceding non - zero elements in column 2 = 2

Offset = 3 + 2 = 5

**Address of (3,3)**

No. of elements in rows 0,1,2 = ( 1+2+3 ) = 6

No. of preceding non - zero elements in row 3 = 3

Offset = 6+3= 9

# Lower triangular matrix - addressing function

- Base address + element size ($\sum_{k=0}^{i} k + j$)

# Symmetric matrix

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 2 | 4 | 6 | 8 | 10 |
| 3 | 6 | 9 | 12 | 15 |
| 4 | 8 | 12 | 16 | 20 |
| 5 | 10 | 15 | 20 | 25 |

# Band Matrix

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 0 | 0 | 0 |
| 3 | 4 | 5 | 0 | 0 |
| 0 | 6 | 7 | 8 | 0 |
| 0 | 0 | 9 | 10 | 11 |
| 0 | 0 | 0 | 12 | 13 |

# Exchange Matrix

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |

# Upper Shift Matrix

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |

# Lower Shift Matrix

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |

# Toeplitz Matrix

- Elements on the same diagonal are same
- Example

| | | | | |
|---|---|---|---|---|
| 10 | 2 | 3 | 4 | 5 |
| 6 | 10 | 2 | 3 | 4 |
| 7 | 6 | 10 | 2 | 3 |
| 8 | 7 | 6 | 10 | 2 |
| 9 | 8 | 7 | 6 | 10 |

Suggest efficient storage representation and write the corresponding addressing function

# Sort an array with elements 0s and 1s

Sample input [1, 0, 0, 1, 1, 0, 1, 0]

Sample output [0, 0, 0, 0, 1, 1, 1, 1]

# Sort an array of 0s, 1s and 2s

- Given an array of size N containing only 0s, 1s, and 2s; sort the array in ascending order.
- **Expected Time Complexity:** O(N)
  **Expected Auxiliary Space:** O(1)