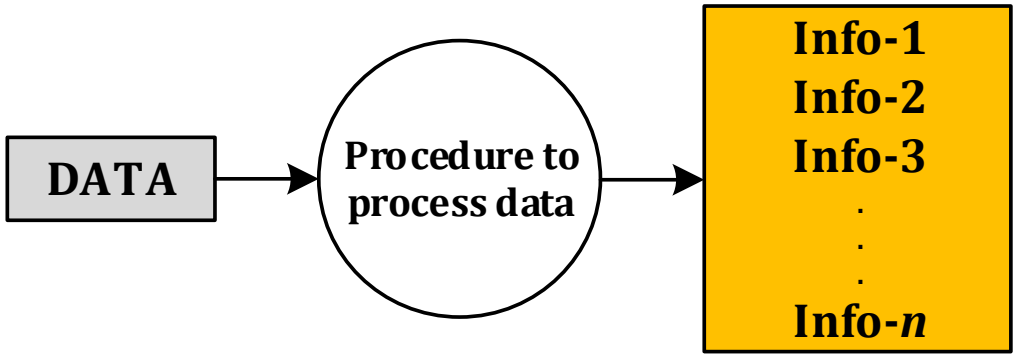
Introduction to Data Structure
&
Array

Subject

IT247 Data Structures And Algorithms

Data & Information

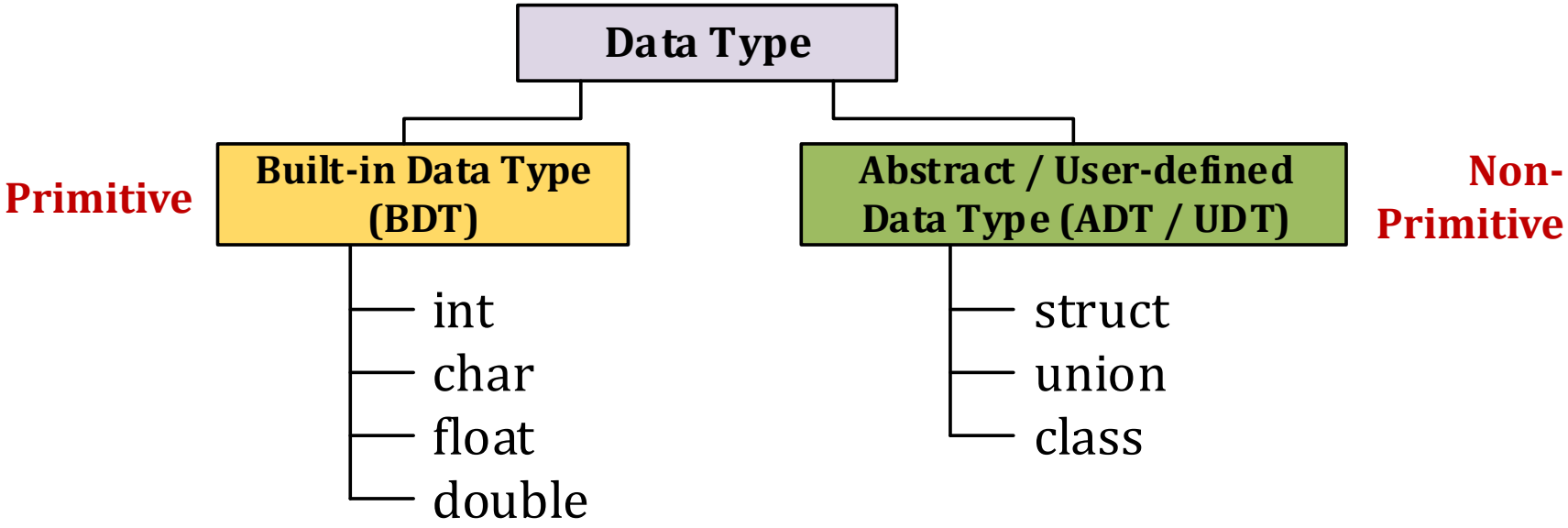
- **Data** means value or a set of values.
  - 35
  - 21/12/2016,
  - "CHARUSAT"
  - 12, 18, 24, 32



- **Information** means meaningful or processed data.
  - 35 — Age of a person
  - 21/12/2016 — Date of Birth
  - "CHARUSAT" — Name of the University
  - 12, 18, 14, 30 — Marks of a subject

Data Type

- **Data type** is a term which refers to the kind of data. That may appear in computation.
  - 35 Numeric (integer)
  - 21/12/2016 Date
  - "CHARUSAT" String
  - 12, 18, 14, 30 Array of integers



What is Data Structure?

- **Data:** Data is set of items
- **Structure:** How to organize data (A particular way of organizing data in computer)

- A data structure is a method for organizing and storing data, which would allow efficient data retrieval and usage.

Why data structure?

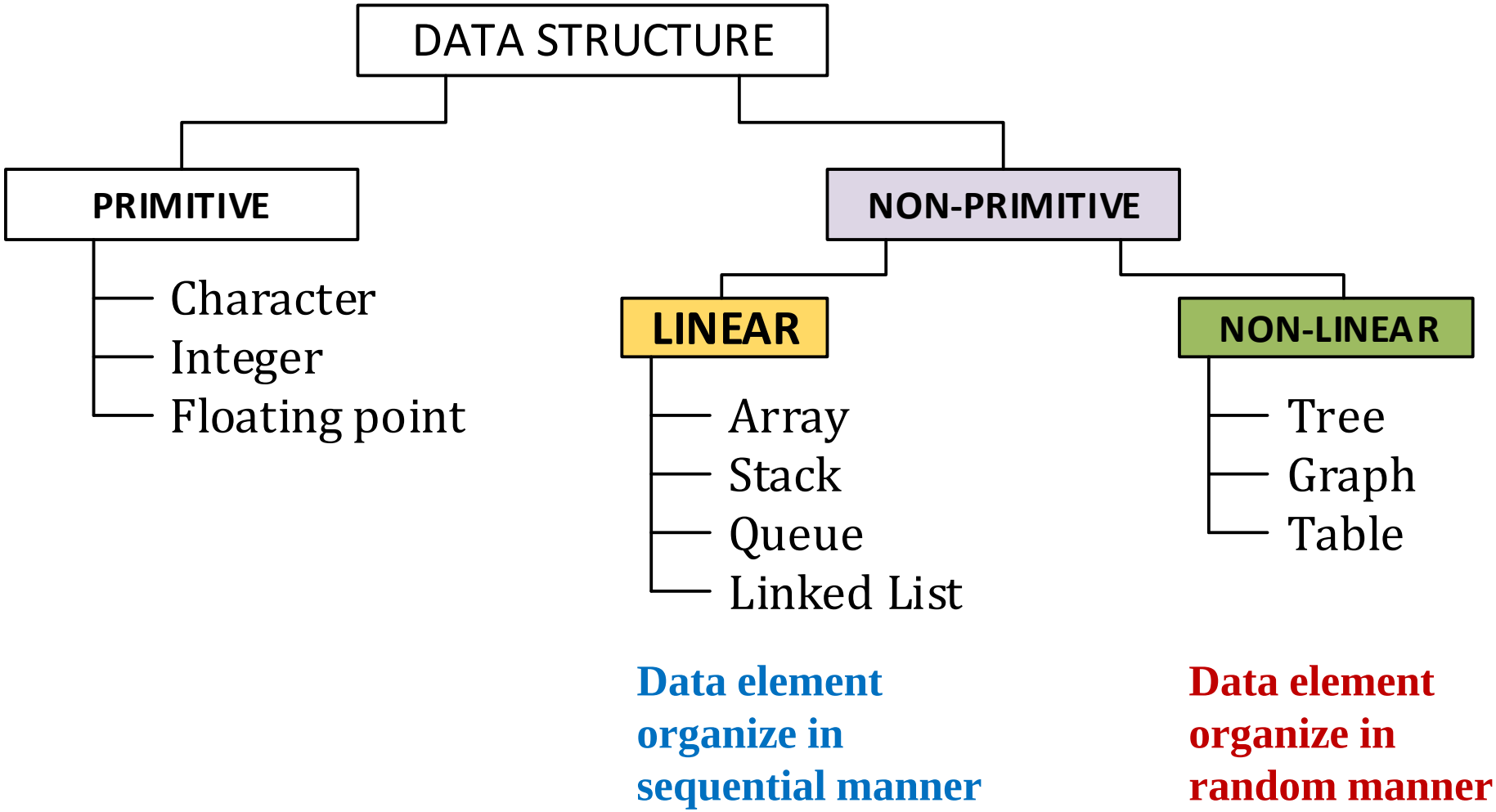
- In computer, manipulation of primitive data does not require any extra effort on the part of user.
- In real-life applications, various kinds of data other than primitive data are involved. So, manipulation of real-life data requires following tasks:

1. Storage representation of user data
2. Retrieval of stored data
3. Transformation of user data

Data structure is used for

- Data Structure is used for
  - How the data should be organized in the memory
  - How the flow of data should be controlled
  - How efficiently it can be retrieved and manipulated
  - How data should be designed and implemented to reduce the complexity and increase the efficiency of the algorithm

Types of Data Structure



Algorithm + Data Structure = Program

- **Algorithm:** Algorithm is a step-by-step finite sequence of instruction, to solve a well defined computational problem

- **Program:** An implementation of an algorithm in some programming language

Array

- Array is one of the linear data structure.
- **Array** is a collection of similar data type variables having contiguous(sequential) memory locations that share a common name.
- Applications:
  1. To implement mathematical vector and matrices, as well as other kinds of rectangular tables: many databases, small & large, consist of 1D arrays whose elements are records.
  2. To implement other data structures, such as heaps, hash tables, queues, stacks
  3. One or more large arrays are sometimes used to emulate in-program dynamic memory allocation, particularly memory pool allocation.

One Dimensional Array

- Notation: **A [*l* : *u*]**

  where, **A** =Array Name

  l=lower bound, *u*=upper bound

- Example:

  if int A[6] declared in C program means

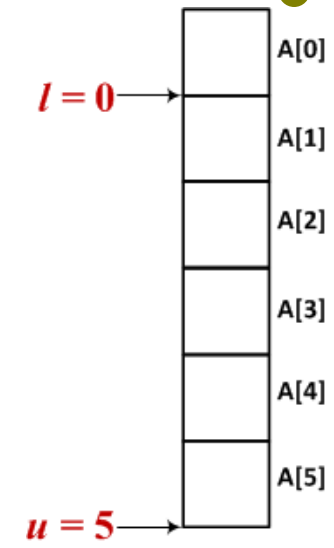
  A[0:5]

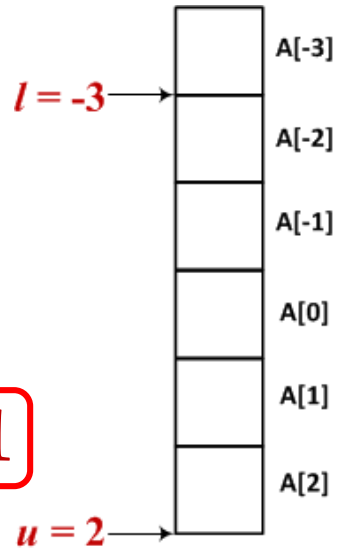
- In general, **A[-3 : 2]** it has 6 elements

  where, *l* =-3

  u=2

- Index of i^{th} element: $\text{Index}(A_i) = \mathbf{L + i - 1}$



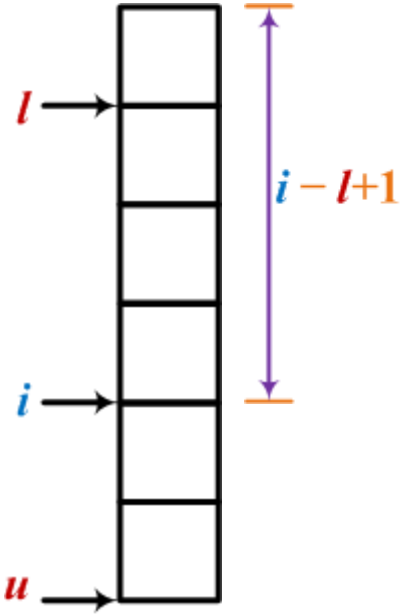


One Dimensional Array

- No of elements: $(u - l + 1)$

- Because elements are stored sequentially in the memory we now derive formula to find the address of an element in one dimension array.

- If we want to find address of A[i]



Address of an element in 1-D

$$\mathbf{A}[\, i \,] = \mathbf{B.A.} + (i - l) \times c$$

where,

i = index

B.A.= Base Address

l = lower bound of an array

c = size of each element in bytes

Example

- A[-2 : 3] (array of int with BA=2001)

  Find Address of A[2].

  No of elements: $(u - l + 1) = (3 - (- 2)+1) = 6$

  $l = -2$, $i = 2$

  $u = 4$,

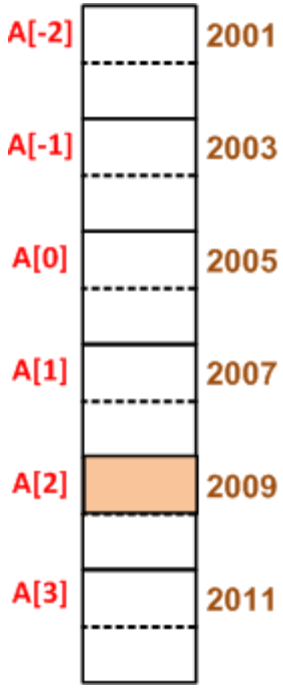
  $c = 4$, BA= 2001

  $\mathbf{A[i] = B.A.+(i - l) \times c}$

  $= 2001+(2 -(-2)) \times 2$

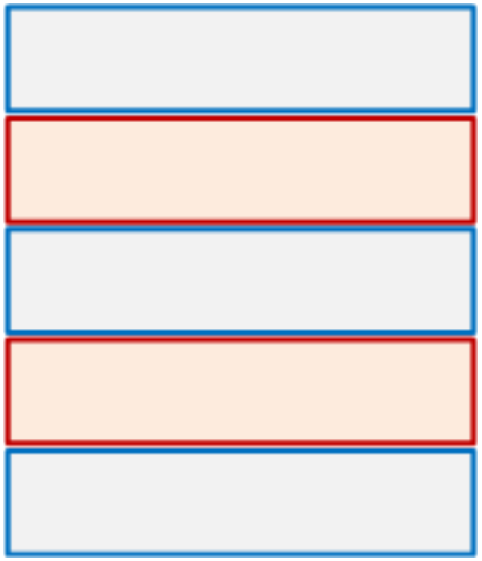
  $= 2001+(4 \times 2)$

  $= \mathbf{2009}$



Two Dimensional Array

- Actually two-dimensional array is one-dimensional array whose each element is one-dimensional array.



Two Dimensional Array

- Notation: $\mathbf{A}[\, l_r : u_r, l_c : u_c \,]$

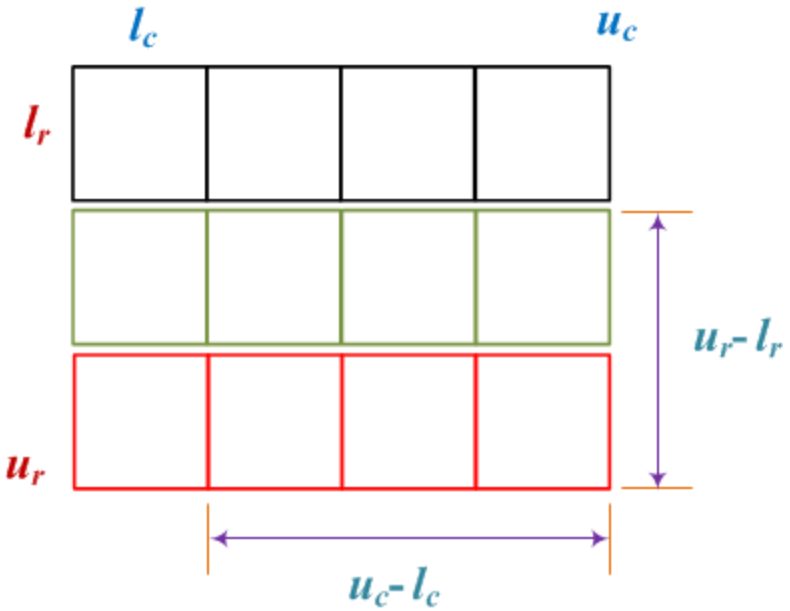
where,

A=Array Name

l_r =lower bound for row, u_r =upper bound for row

l_c =lower bound for col, u_c =upper bound for col

Calculate elements



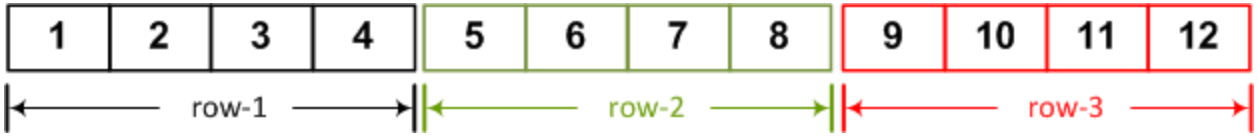
- No of rows: $\mathbf{r} = (u_r - l_r + 1)$
- No of columns: $\mathbf{c} = (u_c - l_c + 1)$
- No of elements: $\mathbf{r} \times \mathbf{c} = (u_r - l_r + 1) \times (u_c - l_c + 1)$

Memory Representation

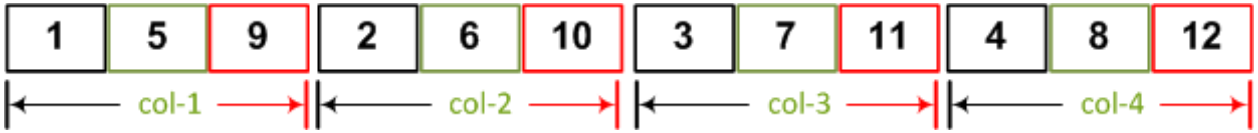
- There are two ways to represent 2-Dimensional array in memory.
- A[**-1 : 1, 2 : 5**] is 3×4 array with 12 elements.

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

1. Row major:



2. Column major:

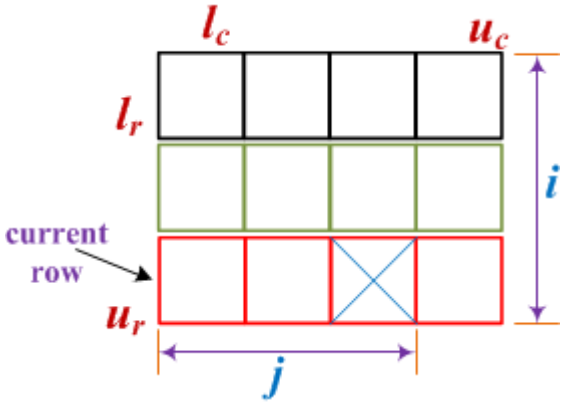


Find the address of an element in 2-D

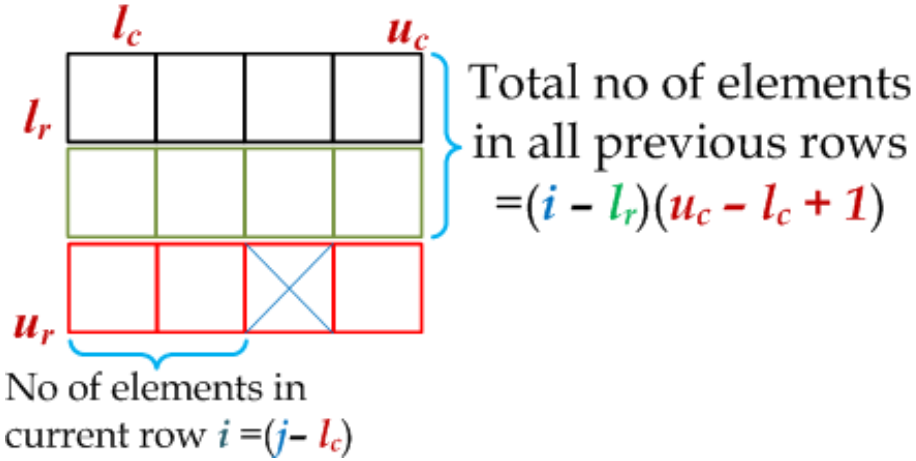
- To find the address of an element **A**[*i*,*j*]:
- First find total no of previous rows/columns from the current.
- One row contains elements=**size of column**
- One column contains elements=**size of row**
- So, we can calculate Total no of elements in all previous rows/columns.
- Second, calculate no of elements in current row/column before current.
- Finally, add first and second multiply by size of an element in bytes gives distance of current element in no of bytes from base address.

Row Major

- To find the address of an element **A**[*i*,*j*]
- First find total no of rows from the current.



- In array, one row contains **c**=$(u_c - l_c + 1)$ elements.



Address of an element

- For Row major:

$$A[i,j] = B.A. + [(i - l_r)(u_c - l_c + 1) + (j - l_c)] \times c$$

- $(i - l_r)$: No of rows before current row i
- $(u_c - l_c + 1)$: No of elements in a row = No of column in an array
- $(j - l_c)$: No of elements before j in current row i
- c: Size of an element in bytes

Total no of elements in all previous rows $=(i - l_r)(u_c - l_c + 1)$

No of elements in current row $i = (j - l_c)$

Example

- If D[-13:1, 4:9] is an array of float find the address of D[-2, 8]. Address of D[-13,4] is 3000.

  Here, l_r =-13, u_r =1, BA=3000, c=4($\because float$)

  l_c =4, u_c =9, i=-2, j=8

  D[-2,8] = **B.A.**+[(i – l_r)(u_c – l_c + 1)+(j– l_c)]×c

  = 3000 + [((-2)-(-13))(9-4+1)+(8-4)] ×4

  = 3000 + [(11×6)+4] ×4

  = 3000 + 280

  = **3280**

Address calculation for Column Major

- If an array is stored as column major representation than the address of an element can be calculated same way.
- First find total columns before current.
- Then, in current column find total elements from the current element.
- Addition of these two gives total no of elements before the current element [*i,j*].
- Multiply with size of element and then add with base address is the answer.

Column Major

$$A[i,j] = B.A. + [(j - l_c)(u_r - l_r + 1) + (i - l_r)] \times c$$

- $(j - l_c)$: No of columns before current column j
- $(u_r - l_r + 1)$: No of elements in a column = No of row in an array
- $(i - l_r)$: No of elements before i in current column j
- c: Size of an element in bytes

- If D[-13:1, 4:9] is an array of float find the address of D[-2, 8]. Address of D[-13,4]=3000.

Here, l_r =-13, u_r =1, BA=3000, c=4($\because$ $float$)

l_c =4, u_c =9, i=-2, j=8

D[-2,8] = $B.A. + [(j - l_c)(u_r - l_r + 1) + (i - l_r)] \times c$

= 3000 + [(8-4)(1-(-13)+1)+(-2-(-13)] ×4

= **3284**

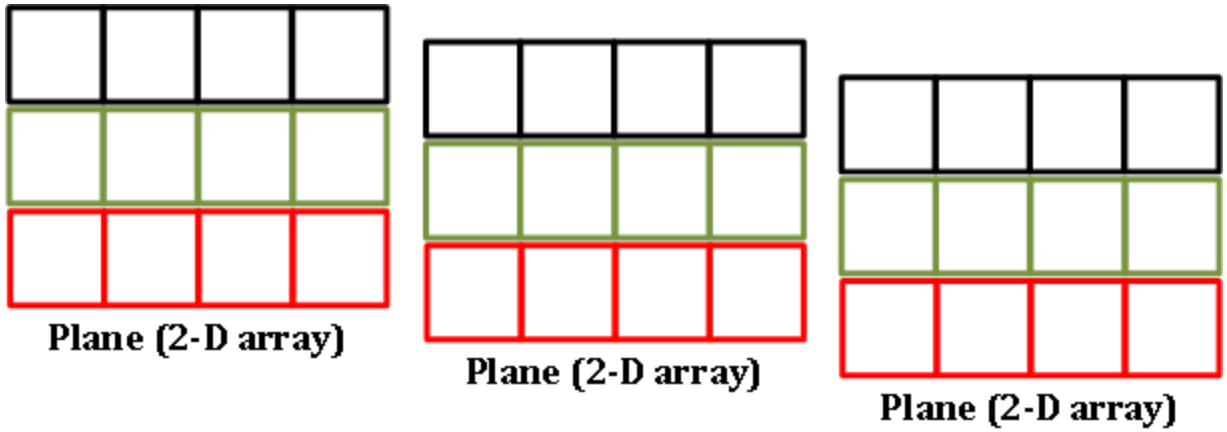
Example

- Given a two dimensional array Zl (3:l0, 10:20) stored in row-major order with base address of 200 and size of each element of 4 bytes, find address of element Z1(5, 15).
- Ans : **Z1[5,15] =308**

Three Dimensional Array

- A three dimensional array is a collection of two dimensional arrays (planes). Each two dimensional array contains rows and columns.
- For example, Array[3][3][4]

No of Planes — No of Rows — No of Columns



Three Dimensional Array

- Notation: **A[** $l_P : u_P$, $l_r : u_r$, $l_c : u_c$**]**

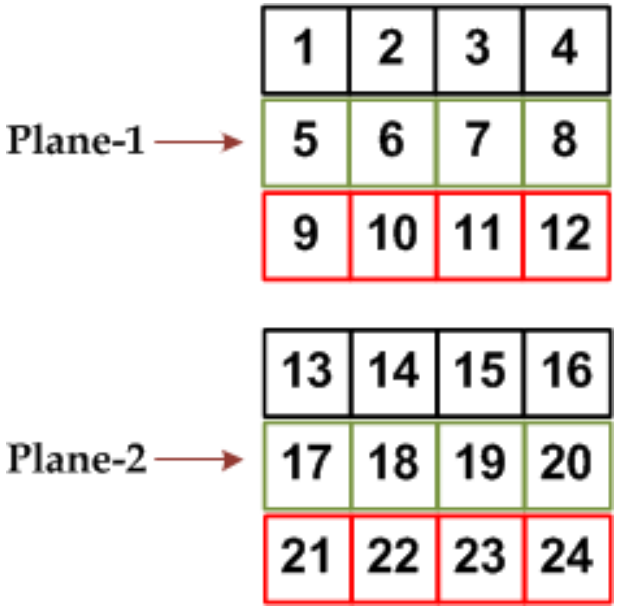
where, **A** =Array Name

l_P =lower bound for plane, u_P =upper bound for plane

l_r =lower bound for row, u_r =upper bound for row

l_c =lower bound for col, u_c =upper bound for col

- A[2][3][4] means two 2-D arrays having size 3×4.
- Here, a 2-D array is denoted as a plane



Three Dimensional Array

- No of Planes: $\mathbf{p} = (u_P - l_P + 1)$
- No of rows: $\mathbf{r} = (u_r - l_r + 1)$
- No of columns: $\mathbf{c} = (u_c - l_c + 1)$
- No of elements: $\mathbf{p} \times \mathbf{r} \times \mathbf{c}$

$$= (u_P - l_P + 1) \times (u_r - l_r + 1) \times (u_c - l_c + 1)$$

Example: **A**[-1:1, 2:4, -10:-6]

No of Planes: (1)-(-1)+1=3

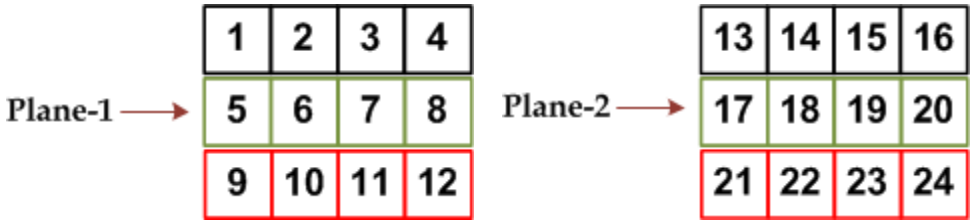
No of rows: (4)-(2)+1=3

No of columns: (-6)-(-10)+1=5

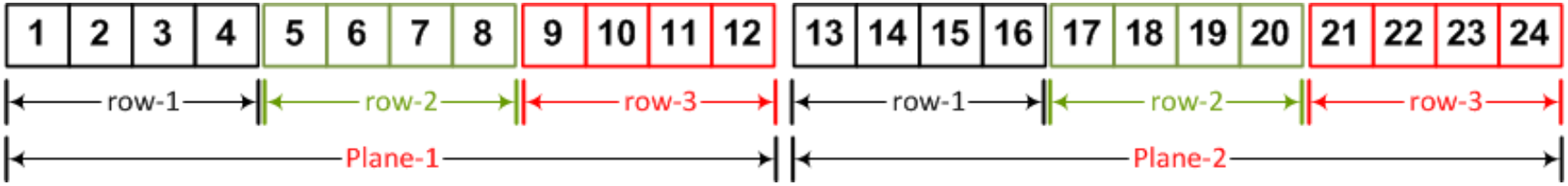
No of elements: 3×3×5=45

Memory Representation

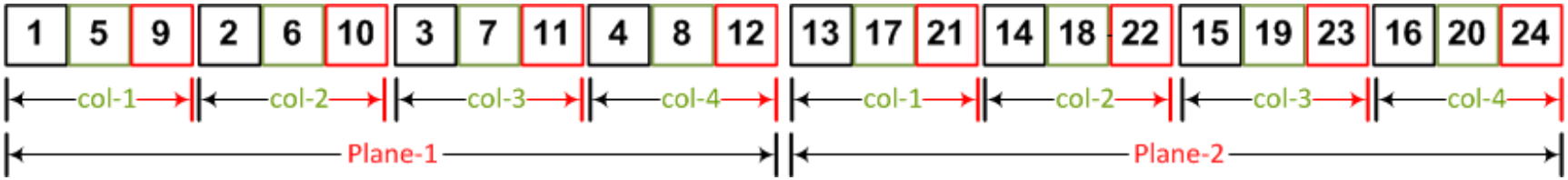
- For multi-dimensional, two types of memory representation is possible.



1. Row major:

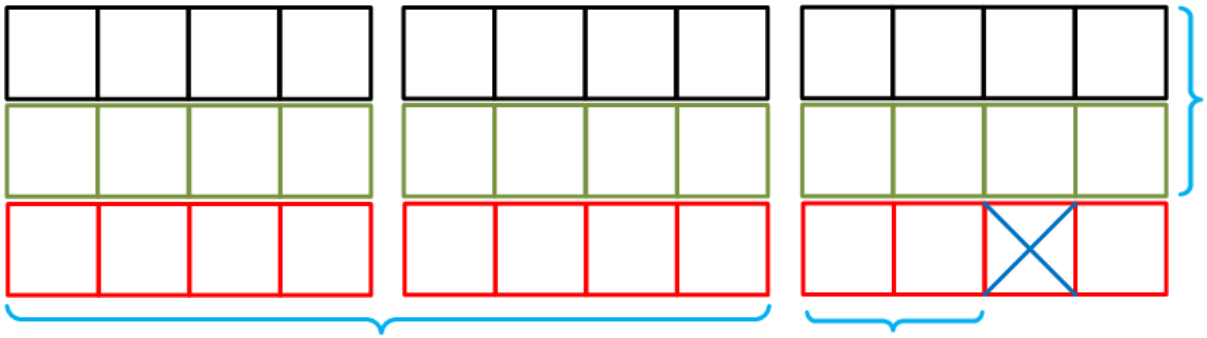


2. Column major:



Find the address of an element in 3-D

- Stored in Row major:



No of elements in all previous rows= $(j - l_r)(u_c - l_c + 1)$

No of elements in all the previous planes= $(i - l_P)(u_r - l_r + 1)(u_c - l_c + 1)$

No of elements in current row= $(k - l_c)$

For Row Major & Column Major

$$A[i,j,k] = B.A. + [(i - l_P)(u_r - l_r + 1)(u_c - l_c + 1) + (j - l_r)(u_c - l_c + 1) + (k - l_c)] \times c$$

- $(i - l_P)$: No of planes before current plane i
- $(u_r - l_r + 1)(u_c - l_c + 1)$: No of elements in a plane = No of elements in an 2D-array
- $(j - l_r)(u_c - l_c + 1) + (k - l_c)$: Same as 2-D array calculation
- c: Size of an element in bytes

$$A[i,j,k] = B.A. + [(i - l_P)(u_r - l_r + 1)(u_c - l_c + 1) + (k - l_c)(u_r - l_r + 1) + (j - l_r)] \times c$$

- $(i - l_P)$: No of planes before current plane i
- $(u_r - l_r + 1)(u_c - l_c + 1)$: No of elements in a plane = No of elements in an 2D-array
- $(k - l_c)(u_r - l_r + 1) + (j - l_r)$: Same as 2-D array column major calculation
- c: Size of an element in bytes

Example

- Find address of B[0,-2,2] if array contains integer with initial address 140. B[-2:0, -4:-1, 1:3]
- Here,

  B.A.=140

  i =0, j =(-2), k=2

  l_P =(-2), u_P = 0

  l_r =(-4), u_r =(-1)

  l_c =1, u_c =3

  c = 2

Example

- Row Major:

$A[i,j,k] = B.A. + [(i - l_P)(u_r - l_r + 1)(u_c - l_c + 1) + (j - l_r)(u_c - l_c + 1) + (k - l_c)] \times c$

$= 140 + [(0-(-2))((-1)-(-4)+1)(3-1+1) + ((-2)-(-4))(3-1+1) + (2-1)] \times 2$

$= 140 + [(2)(4)(3) + (2)(3) + 1] \times 2$

$= 140 + [31] \times 2 = \mathbf{202}$

- Column Major:

$A[i,j,k] = B.A. + [(i - l_P)(u_r - l_r + 1)(u_c - l_c + 1) + (k - l_c)(u_r - l_r + 1) + (j - l_r)] \times c$

$= 140 + [(0-(-2))((-1)-(-4)+1)(3-1+1) + (2-1))((-1)-(-4)+1) + ((-2)-(-4)] \times 2$

$= 140 + [(2)(4)(3) + (1)(4) + 2] \times 2$

$= 140 + [30] \times 2 = \mathbf{200}$

Example

- For a given 3-D array A[-2:2, 1:4, 6:9] with base address 2000 and size of element is 4 bytes. Find out total number of elements and address of A[1, 3, 8] element in column major order.
- Ans : A[1, 3, 8] = 2232

- For given array Z[-4:-1, 10:13, -1:1] find the total number of elements. Assume that the base address is 5001 and each element required 2 bytes. Find address of Z[-2,12,0] element if it is stored in (a) row major order (b) column major.