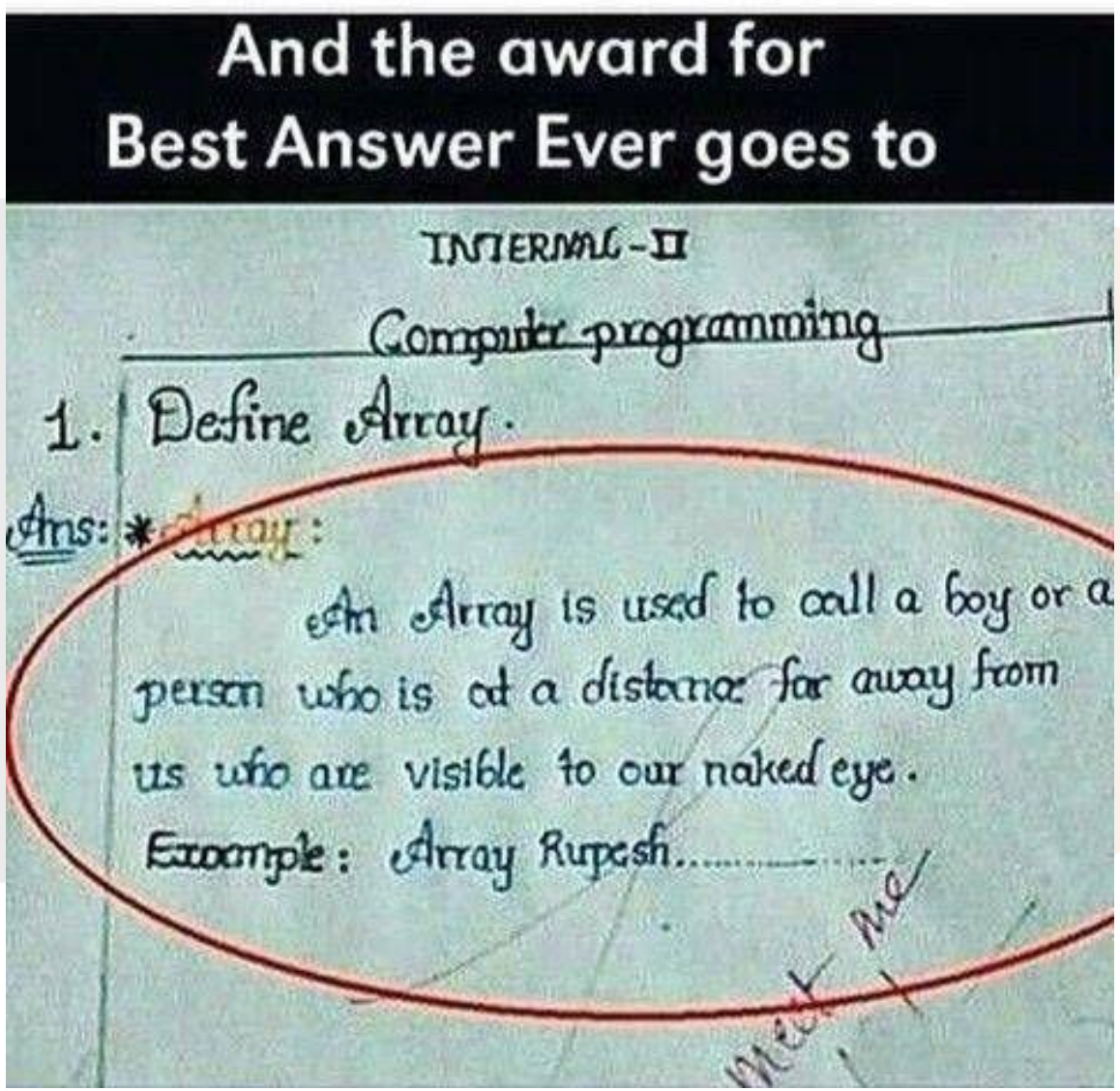
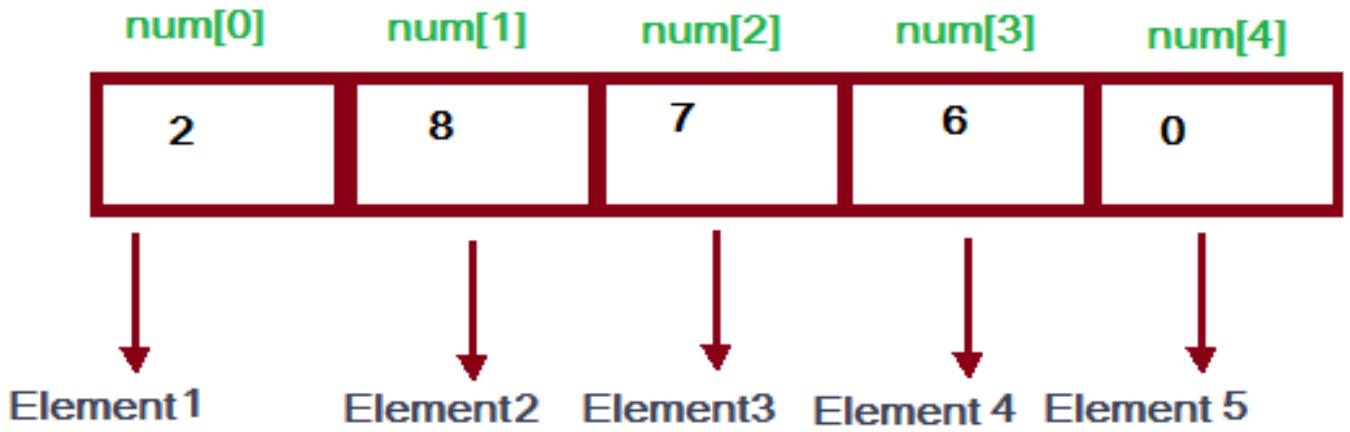
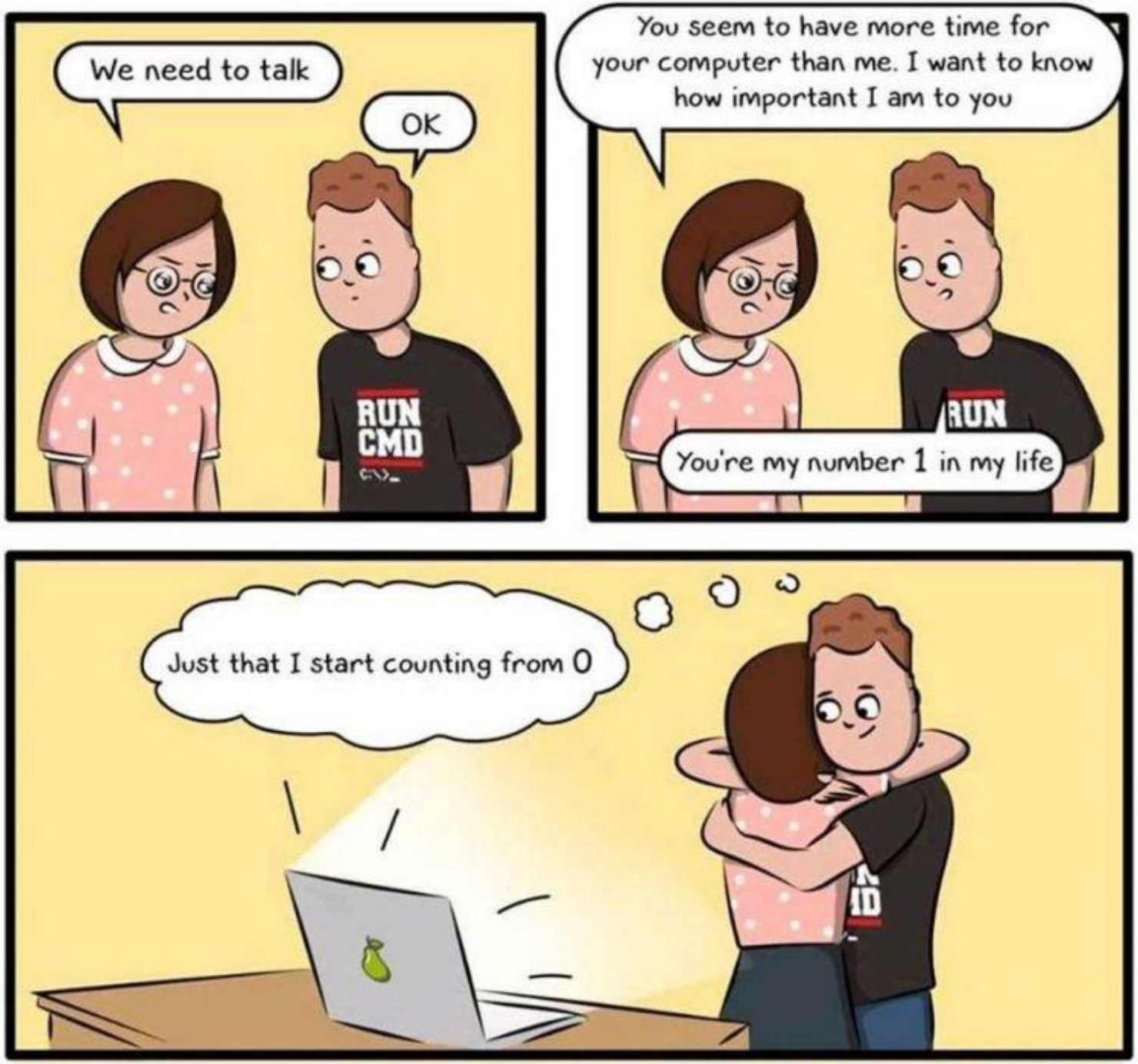
Array



Array

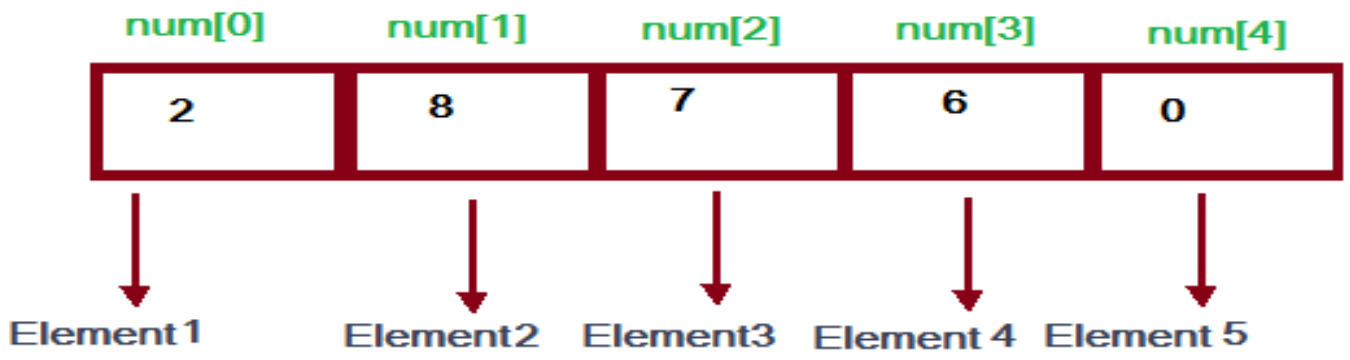
- An array is a collection of items stored at contiguous memory locations.
- The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).
- Each element can be uniquely identified by their index in the array.





How to declare an array in C

- datatype array Name[array Size];
- int data[100];



How to initialize an array

- It is possible to initialize an array during declaration. For example,
  - int mark[5] = {19, 10, 8, 17, 9};
- You can also initialize an array like this.
  - int mark[] = {19, 10, 8, 17, 9};
- Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

- Change Value of Array elements
  - int mark[5] = {19, 10, 8, 17, 9};
  - make the value of the third element to -1
    - mark[2] = -1;
  - make the value of the fifth element to 0
    - mark[4] = 0;

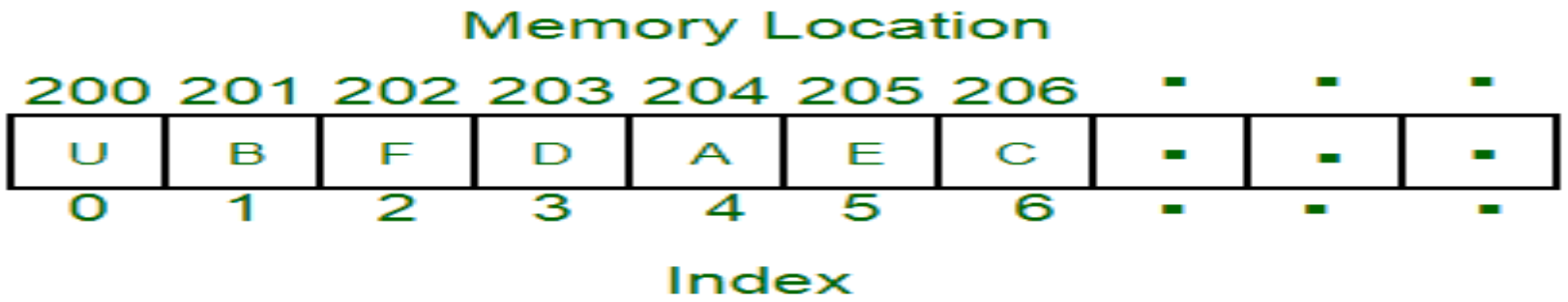
- Types of indexing in array:
  - 0 (zero-based indexing): The first element of the array is indexed by subscript of 0
  - 1 (one-based indexing): The first element of the array is indexed by subscript of 1
  - n (n-based indexing): The base index of an array can be freely chosen. Usually programming languages allowing n-based indexing also allow negative index values and other scalar data types like enumerations, or characters may be used as an array index.

Break

Size of an array

- Number of elements = (Upper bound – Lower Bound) + 1
  - Lower bound index of the first element of the array
  - Upper bound index of the last element of the array

- Size = number of elements * Size of each elements in bytes



One Dimensional array

- Address of the element at k^{th} index
  - a[k] =

One Dimensional array

- Address of the element at k^{th} index
  - a[k] = B + W*k
  - a[k] = B + W*(k – Lower bound)
    - B is the base address of the array
    - W is the size of each element
    - K is the index of the element
    - Lower bound index of the first element of the array
    - Upper bound index of the last element of the array

Q Let the base address of the first element of the array is 250 and each element of the array occupies 3 bytes in the memory, then address of the fifth element of a one- dimensional array a[10] ?

Q An array has been declared as follows

A: array [-6--------6] of elements where every element takes 4 bytes, if the base address of the array is 3500 find the address of array[0]?

Q A program P reads in 500 integers in the range [0...100] experimenting the scores of 500 students. It then prints the frequency of each score above 50. What would be the best way for P to store the frequencies? **(GATE - 2005) (2 Marks)**

(A) An array of 50 numbers

(B) An array of 100 numbers

(C) An array of 500 numbers

(D) A dynamically allocated array of 550 numbers

Break

Two-Dimensional array

- The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns.

- However, 2D arrays are created to implement a relational database look a like data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Two-Dimensional array

```
data_type array_name[rows][columns];

int disp[2][4] = {
    {10, 11, 12, 13},
    {14, 15, 16, 17}
        };
```

OR

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

Implementation of 2D array

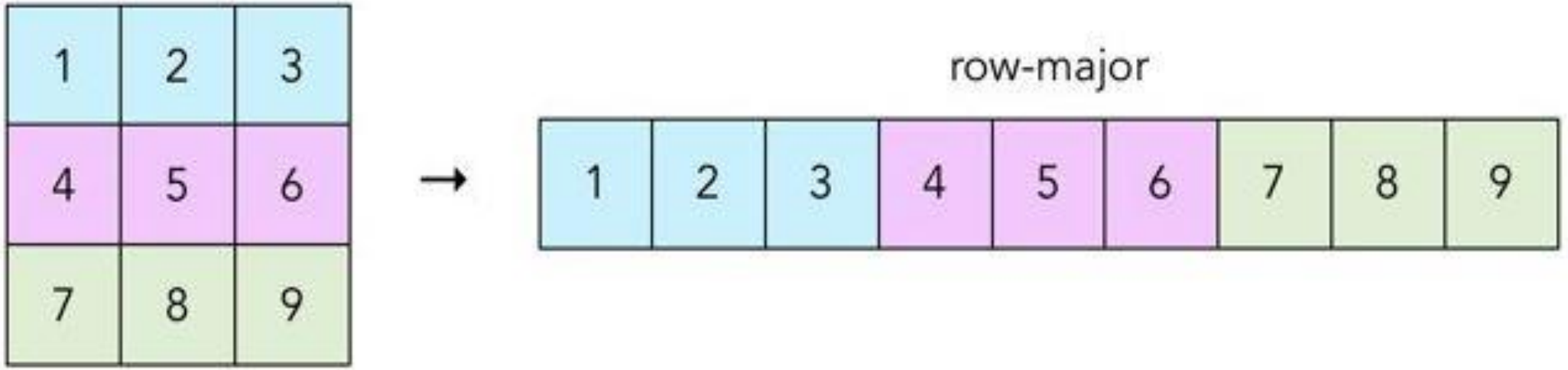
- In computing, row-major order and column-major order are methods for storing multidimensional arrays in linear storage such as random access memory.

- The difference between the orders lies in which elements of an array are contiguous in memory.

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

Row Major implementation of 2D array

- In Row major method elements of an array are arranged sequentially row by row.
- Thus, elements of first row occupies first set of memory locations reserved for the array, elements of second row occupies the next set of memory and so on.



Row Major implementation of 2D array

Address of a[i][j] =
B = Base address
W = Size of each element
L_1 = Lower bound of rows
U_1 = Upper bound of rows
L_2 = Lower bound of columns
U_2 = Upper bound of columns

Row Major implementation of 2D array

Address of a[i][j] = B + W*[(U_2-L_2+1) $(i-L_1)$ + $(j-L_2)$]

B = Base address

W = Size of each element

L_1 = Lower bound of rows

U_1 = Upper bound of rows

L_2 = Lower bound of columns

U_2 = Upper bound of columns

(U_2-L_2+1) = numbers of columns

$(i-L_1)$ = number of rows before us

$(j-L_2)$ = number of elements before us in current row

Q Let A be a two dimensional array declared as follows:

A: array [1 ... 10] [1 ... 15] of integer;

Assuming that each integer takes one memory location, the array is stored in row-major order and the first element of the array is stored at location 100, what is the address of the element a[i][j] ? **(Gate-1998) (2 Marks)**

(A) 15i+ j+ 84

(B) 15j+ i+ 84

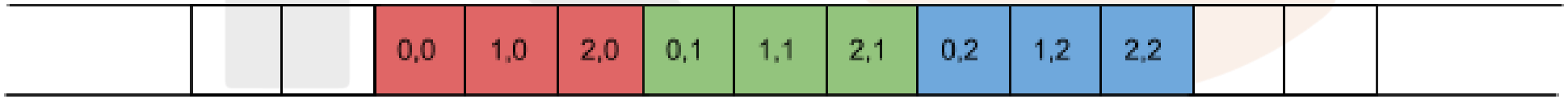
(C) 10i+ j+ 89

(D) 10j+ i+ 89

Break

Column Major implementation of 2D array

- In Column major method elements of an array are arranged sequentially column by column. Thus, elements of first column occupies first set of memory locations reserved for the array, elements of second column occupies the next set of memory and so on.



Column Major implementation of 2D array

Address of a[i][j] =
B = Base address
W = Size of each element
L_1 = Lower bound of rows
U_1 = Upper bound of rows
L_2 = Lower bound of columns
U_2 = Upper bound of columns

Column Major implementation of 2D array

Address of a[i][j] = B + W*[(U_1-L_1+1) (j-L_2) + (i-L_1)]

B = Base address

W = Size of each element

L_1 = Lower bound of rows

U_1 = Upper bound of rows

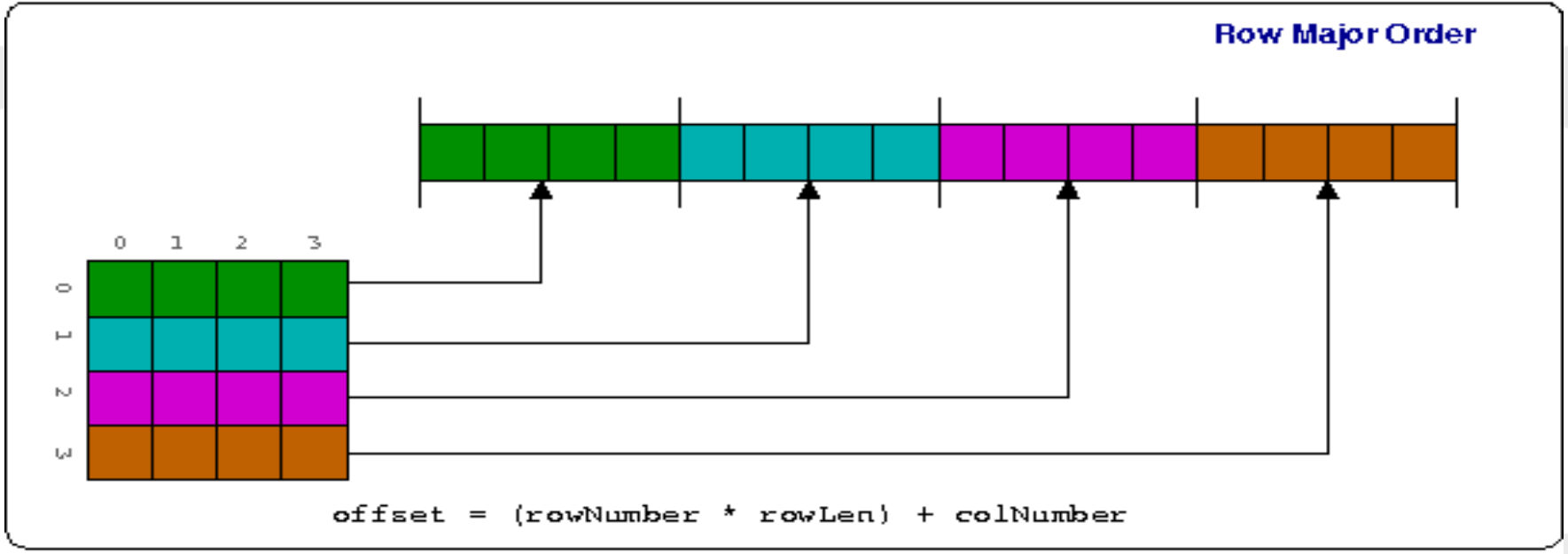
L_2 = Lower bound of columns

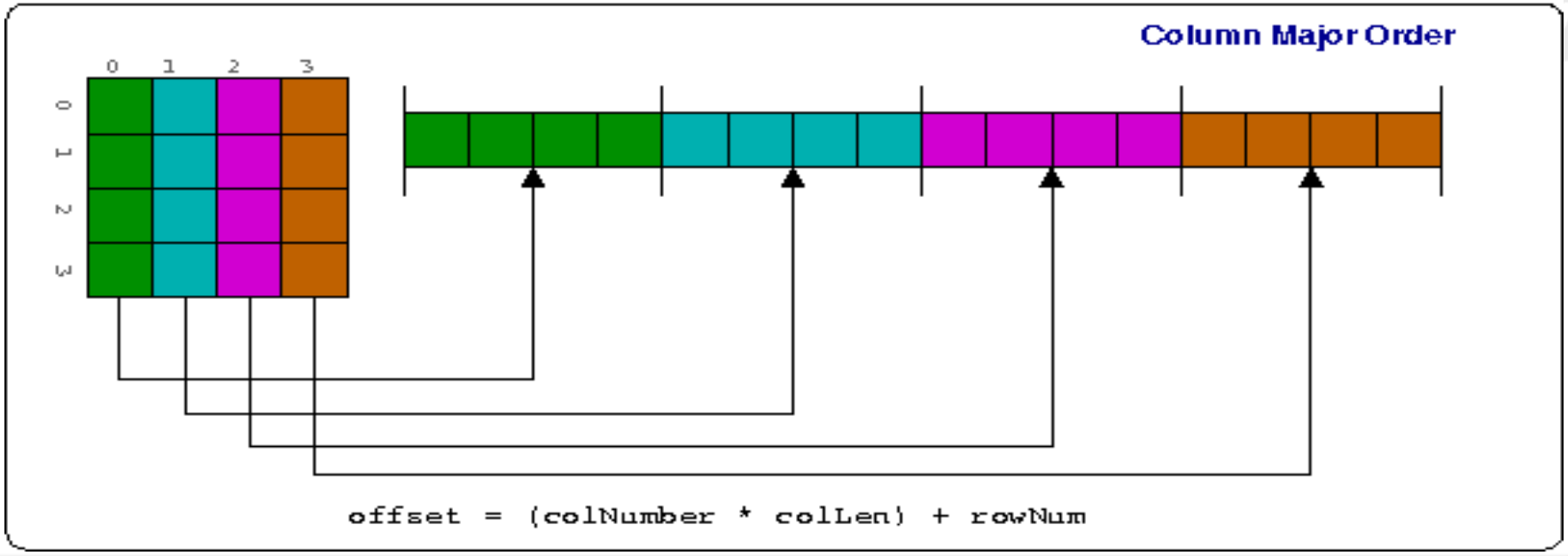
U_2 = Upper bound of columns

(U_1-L_1+1) = numbers of rows

(j-L_2) = number of columns before us

(i-L_1) = number of elements before us in current column





Q An array VAL[1…15][1…10] is stored in the memory with each element requiring 4 bytes of storage. If the base address of the array VAL is 1500, determine the location of VAL[12][9] when the array VAL is stored

(i) Row wise

(ii) Column wise.

Q A[5.....15,-8.....8] is A[11][17] and we are supposed to find A[8][5] - Row Major Order Base Address:800, each element occupies 4 memory cells?

Q Two matrices M_1 and M_2 are to be stored in arrays A and B respectively. Each array can be stored either in row-major or column-major order in contiguous memory locations. The time complexity of an algorithm to compute $M_1 \times M_2$ will be **(Gate-2004) (2 Marks)**

(A) best if A is in row-major, and B is in column- major order

(B) best if both are in row-major order

(C) best if both are in column-major order

(D) independent of the storage scheme

Q An n x n array v is defined as follows:

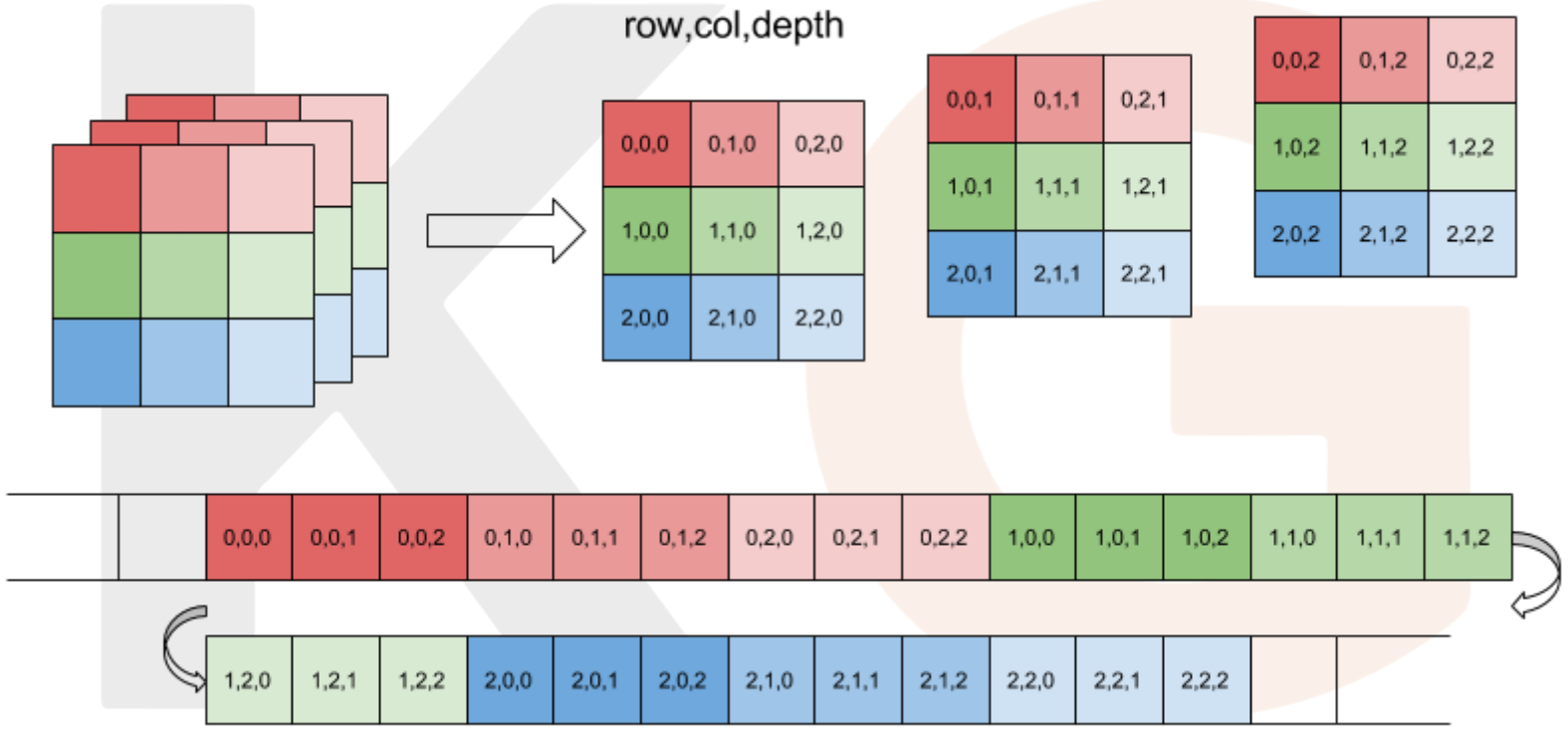
v[i, j] = i-j for all i, j, 1 <= i <= n, 1 <= j <= n

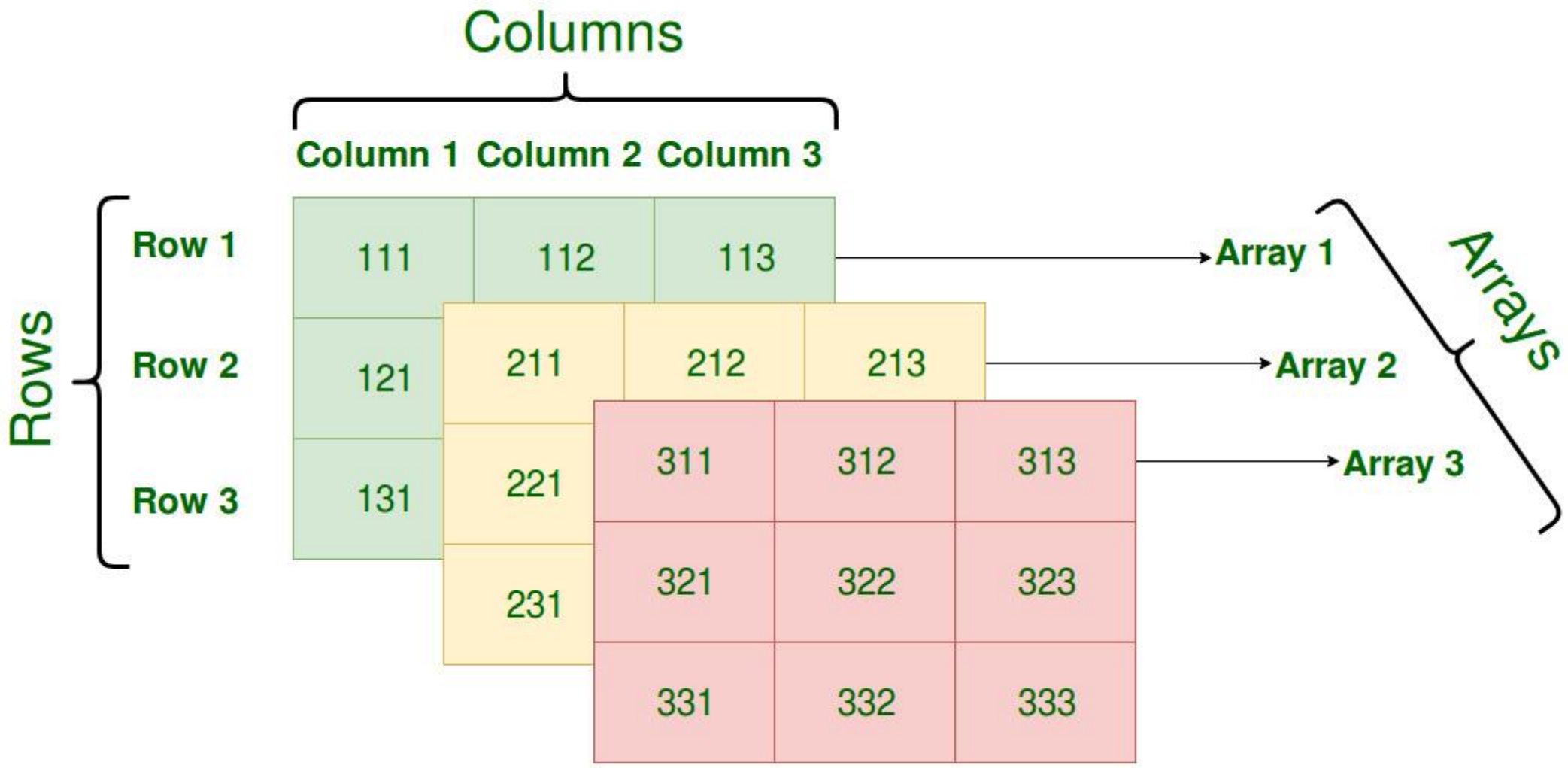
The sum of the elements of the array v is **(Gate-2000) (1 Marks)**

(A) 0 **(B)** n-1 **(C)** $n^2 - 3n + 2$ **(D)** $n^2 (n+1)/2$

Break

3-Dimensional array





Break

N-Dimensional array

$A([L_1]---[U_1]), ([L_2]---[U_2]), ([L_3]---[U_3]), ([L_4]---[U_4])-------([L_N]---[U_N])$

Location of A [I, j, k, ----, x] =

N-Dimensional array

$A([L_1]---[U_1]), ([L_2]---[U_2]), ([L_3]---[U_3]), ([L_4]---[U_4])-------([L_N]---[U_N])$

Location of A [I, j, k, ----, x] =

$$B + (i-L_1)(U_2-L_2+1)(U_3-L_3+1)(U_4-L_4+1)\text{ ----}(U_n-L_n+1)$$
$$+ (j-L_2)(U_3-L_3+1)(U_4-L_4+1)\text{ ----}(U_n-L_n+1)$$
$$+ (k-L_3)(U_4-L_4+1)\text{ ----}(U_n-L_n+1)$$
$$+$$
$$+$$
$$+$$
$$+ (x-L_n)$$

Break

Q A Young tableau is a 2D array of integers increasing from left to right and from top to bottom. Any unfilled entries are marked with ∞, and hence there cannot be any entry to the right of, or below a ∞. The following Young tableau consists of unique entries **(GATE - 2015) (2 Marks)**

1	2	5	14
3	4	6	23
10	12	18	25
31	∞	∞	∞

When an element is removed from a Young tableau, other elements should be moved into its place so that the resulting table is still a Young tableau (unfilled entries may be filled in with a ∞). The minimum number of entries (other than 1) to be shifted, to remove 1 from the given Young tableau is _____

(A) 2 **(B)** 5 **(C)** 6 **(D)** 18

Q Let A be a square matrix of size n x n. Consider the following program. What is the expected output? **(GATE - 2014) (1 Marks)**

```
C = 100
for i = 1 to n do
{
   for j = 1 to n do
   {
      Temp = A[i][j] + C
      A[i][j] = A[j][i]
      A[j][i] = Temp - C
   }
}
for i = 1 to n do
   for j = 1 to n do
      Output(A[i][j]);
```

(A) The matrix A itself
(B) Transpose of matrix A
(C) Adding 100 to the upper diagonal elements and subtracting 100 from diagonal elements of A
(D) None of the above

Q Suppose you are given an array s[1..n] and a procedure reverse (s, i, j) which reverses the order of elements in a between positions i and j (both inclusive). What does the following sequence do, **(GATE - 2014) (1 Marks)**

where 1 <= k < n:

```
reverse(s, 1, k) ;
reverse(s, k + 1, n);
reverse(s, 1, n);
```

(A) Rotates s left by k positions

(B) Leaves s unchanged

(C) Reverses all elements of s

(D) None of the above

Q Consider a n*n square matrix A, such that

A[i][j] = 0 if(i<j)

a) find the space required to store the array in memory

b) derive an address access formula to access this matrix (if stored in row major order)

Break

Q In a compact one dimensional array representation for lower triangular matrix (all elements above diagonal are zero) of size n x n, non zero elements of each row are stored one after another, (i.e. elements of lower triangle) of each row are stored one after another, starting from first row, the index of $(i, j)^{th}$ element of the lower triangular matrix in this new representation is **(GATE - 1994) (2 Marks)**

(A) i+j **(B)** i + j-1 **(C)** j-1 + i(i-1)/2 **(D)** i+j(j-1)/2

Q Consider a n*n square matrix A, such that

A[i][j] = 0 if(i<j)

a) derive an address access formula to access this matrix (if stored in column major order)

Q Consider a n*n square matrix A, such that
A[i][j] = A[j][i]
find the space required to store the array in memory

Q Consider a n*n square matrix A, such that

A[i][j] = 0 if |i-j| > 1

find the space required to store the array in memory

Q Consider a n*n square matrix A, such that

A[i][j] = A[i-1][j-1] if i>0 && j>0, 0<=i,j<=n-1

find the space required to store the array in memory

Q Consider a n*n square matrix A, such that

	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33