Two-Dimensional Array

- 2D array can be defined as an array of arrays.
- The 2D array is organized as matrices which can be represented as a collection of rows and columns.
- It provides ease of holding the bulk of data at once

Initialization of 2D-Arrays

- int arr[2][2] = {0,1,2,3}; (compile time allocation)
- int arr[2][3]; (run time allocation)

Memory Allocation of 2D
1. Row Major ordering

- All the rows of the 2D array are stored into the memory contiguously.

	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)
2	(2,0)	(2,1)	(2,2)

(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)

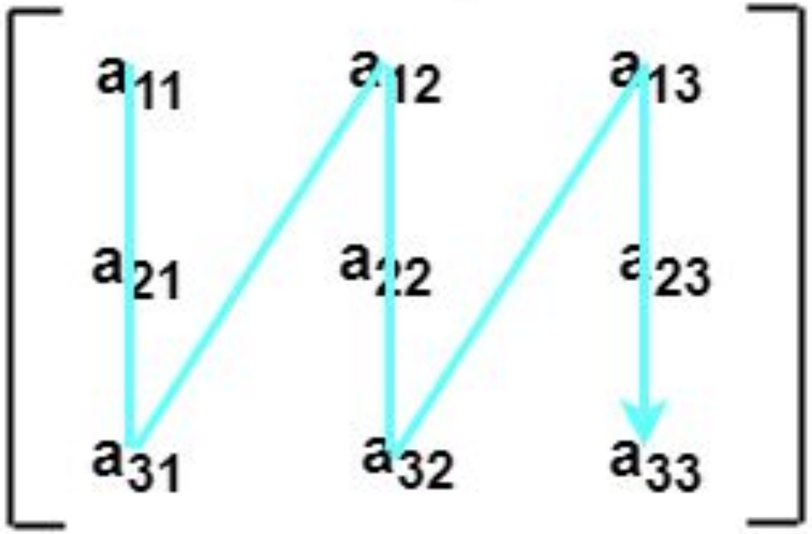
first, the 1st row of the array is stored in the memory completely, then the 2nd row of the array is stored in the memory completely and so on till the last row.

2. Column Major ordering

- all the columns of the 2D array are stored in the memory

	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)
2	(2,0)	(2,1)	(2,2)

(0,0)	(1,0)	(2,0)	(0,1)	(1,1)	(2,1)	(0,2)	(1,2)	(2,2)



first, the 1st column of the array is stored in the memory completely, then the 2nd row of the array is stored in the memory completely and so on till the last column of the array.

EXAMPLE:

1	**2**	**3**	**4**	**5**	**6**

100 104 108 112 116 120 124

1	**4**	**2**	**5**	**3**	**6**

100	**104**	**108**
1	**2**	**3**
a[0][0]	**a[0][1]**	**a[0][2]**
4	**5**	**6**
a[1][0]	**a[1][1]**	**a[1][2]**

Row major

Column major

112	**116**	**120**

Row major ordering A[Lr- – – – – – Ur, Lc- – – – – – Uc]

Address of A[I][J] = B + W * ((I – LR) * N + (J – LC))

I = Row Subset of an element whose address is to be found,

J = Column Subset of an element whose address is to be found,

B = Base address,

W = Storage size of one element stored in an array(in byte),

LR = Lower Limit of row/start row index of the matrix(If not given assume it as zero),

LC = Lower Limit of column/start column index of the matrix(If not given assume it as zero),

N = Number of columns given in the matrix.

N=Number of columns (N) will be calculated as = (Uc – Lc) + 1

Example

A[Lr- – – – – – Ur, Lc- – – – – – Uc]

Given an array, **arr[1………10][1………15]** with base value **100**, and the size of each element is **1 Byte** in memory. Find the address of **arr[8][6]** with the help of row-major order

Row-major order

*Address of A[I][J] = B + W * ((I – LR) * N + (J – LC))*

B=100

W=1

I=8

LR=1

N= *(Uc – Lc) + 1 =15-1+1=15*

J=6

LC=1

□ *100+1((8-1)(15)+(6-1))* □ *=100+110=210*

Address of A[I][J] = 210

Column major Ordering **A[Lr- – – – – – Ur, Lc- – – – – – Uc]**

Address of A[I][J] = B + W * ((J – LC) * M + (I – LR))

I = Row Subset of an element whose address is to be found,

J = Column Subset of an element whose address is to be found,

B = Base address,

W = Storage size of one element stored in an array(in byte),

LR = Lower Limit of row/start row index of the matrix(If not given assume it as zero),

LC = Lower Limit of column/start column index of the matrix(If not given assume it as zero),

M = Number of rows given in the matrix.

M=Number of rows (M) will be calculated as = (Ur – Lr) + 1

Example A[Lr- – – – – – Ur, Lc- – – – – – Uc]

Given an array, **arr[1………10][1………15]** with base value **100**, and the size of each element is **1 Byte** in memory. Find the address of **arr[8][6]** with the help of Column major order

Example A[Lr- – – – – – Ur, Lc- – – – – – Uc]

Given an array, **arr[1………10][1………15]** with base value **100**, and the size of each element is **1 Byte** in memory. Find the address of **arr[8][6]** with the help of Column major order

Address of A[I][J] = B + W * ((J – LC) * M + (I – LR))

B=100

W=1

J=6

LR=1

M= *(Ur – Lr) + 1 =10-1+1=10*

I=8

LC=1

□ *100+1((6-1)(10)+(8-1))* □ *=100+57=157*

Address of A[I][J] = 157

Example 2:

A 2-D array A[4....7, -1....3] requires 2 bytes of storage space for each element. If the array is stored in row-major from having base address 100, then the address of A[6, 2] will be

______

Example 2:

A 2-D array A[4....7, -1....3] requires 2 bytes of storage space for each element. If the array is stored in row-major from having base address 100, then the address of A[6, 2] will be

______

ANS:126

Exercise

An array X [-15……….10, 15……………40] requires one byte of storage. If beginning location is 1500 determine the location of X [5][20].

lbr = , ubr = , lbc = , ubc =
No. of rows =
No of columns =
Size =
BA =

Row Major Wise Calculation of above equation

Address of A [i][j] = BA + [(i-lbr) * column_size + (j-lbc)] *size

Column Major Wise Calculation of above equation

A[i][j] = BA + [(j-lbc) * row_size + (i-lbr)] *size

Exercise

An array X [-15……….10, 15……………40] requires one byte of storage. If beginning location is 1500 determine the location of X [5][20].

lbr = -15, ubr = 10, lbc = 15, ubc = 40
No. of rows = ubr - lbr + 1 = 10 - (-15) +1=26,
No of columns = ubc - lbc + 1 = 40 - 15 + 1 = 26
Size = 1
BA = 1500

Row Major Wise Calculation of above equation

Address of A [i][j] = BA + [(i-lbr) * column_size + (j-lbc)] *size

Address of X [5][20] = 1500 + [(5 – (-15))*26 + (20 – 15)] * 1

= 1500 + [20 * 26 + 5] * 1 = 1500 + [520 + 5] * 1

= 1500 + 525

= 2025 **[Ans]**

Column Major Wise Calculation of above equation

A[i][j] = BA + [(j-lbc) * row_size + (i-lbr)] *size

Address of X [5][20] = 1500 + [(20 – 15) * 26 + (5 - (-15))] * 1

= 1500 + [5 * 26 + 20] * 1

= 1500 + [150] * 1

= 1650 **[Ans]**

Advantages of using arrays:

- Arrays allow random access to elements. This makes accessing elements by position faster.
- Arrays represent multiple data items of the same type using a single name

Disadvantages of using arrays:

- once the size of an array is declared it cannot be changed because of static memory allocation.
- Here Insertion(s) and deletion(s) are difficult as the elements are stored in consecutive memory locations and the shifting operation is costly too.

Application of Array

- Used in solving matrix problems.
- Applied as a lookup table on a computer.
- Databases records are also implemented by the array.
- Helps in implementing sorting algorithm.
- Arrays can be used for CPU scheduling.
- Used to Implement other data structures like Stacks, Queues, Heaps, Hash tables, etc.