Arrays, Linked Lists, Stacks, Queue, AVL Trees, Binary Trees, Binary Search Trees, Scoping

Date: 25/5/13 Page no. 164

Lecture 13

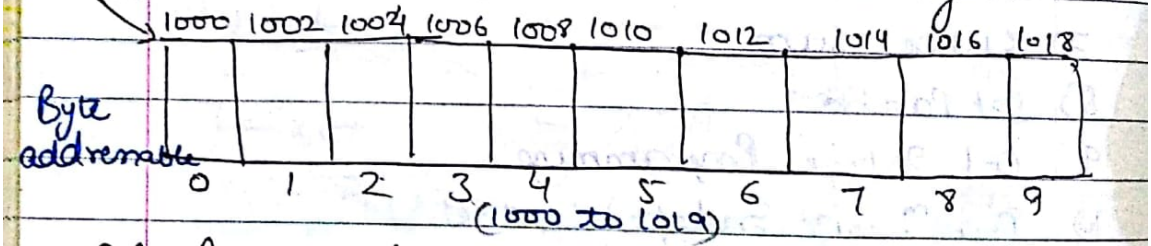
ARRAYS

int a, b, c, d, e; // 5 variables & stored in M/M (may/may not be continous)

int a[5]; // always continous M/M allocation.

1-D Array

int a[10]; // 2×10 = 20 Bytes



QA. Array starts from 0 why not 1?

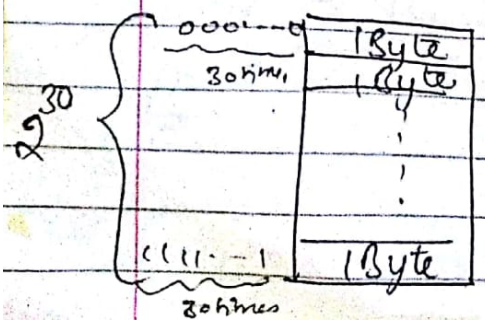
Ans: To avoid offset calculation

1 Byte addres - Byte addressable

1 word address - word "

* Address line = 30 bits

Data lines = 8 bits 2^{30} addresses are possible



$2^{30} * 1$-Byte = 1 GB

☆ 25 × 8 (25 → Address line, 8 → Data lines)

2^{25} rows & every row contains 8 bits.

$2^{25} * 1$ Byte = 32 MB

☆ 30 × 8 (30 → Address lines, 8 → Data lines)

For every 8 bits one address is present.

$2^{30} * 1$ Byte = 1GB — i.e. it is Byte addressable.

☆ 25 × 1 word (1 word → 32 bits ⇒ 4 Bytes)

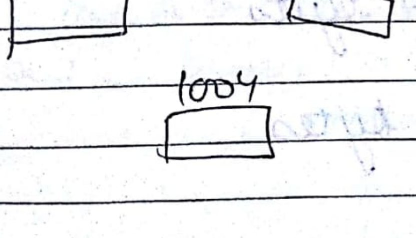
2^{25} rows & every row contains 32 bits data.

$2^{25} \times 1$ word = 32 M word.

* 1000 (address of Byte) = 1 Byte data will come } Byte addressable

* 1000 (address of word) = 1 word data will come } Word addressable

☆ Another way of array:-



= ∵ Continous m/m locations

Select
Date: 25/5/13
Page no. 166

⇒ Declaration is to create the Memory

Ex int a[10];

↓ data type required to create m/m of how many bytes.

→ char a[10]; // 10 Bytes m/m allocated Declaration

→ a[10]; // 10th element accessing

→ int *p; // Declaration (saying p is a pointer) (Sin p will store integer address) & creates m/m

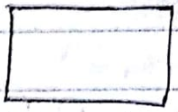
*p; // using the variable. (value at p)

Let address takes 4 bytes address

* int *p; ⇒ Integer variable is stored in p

4 bytes

p



* char *p; ⇒ Character variable address is stored in p

4 bytes

* float p; // 4 bytes

float *p; // 4 bytes

Date: 25/5/13
Page no. 167

★ int a[10] = {10, 11. - - - .19}; Declaration followed by Initialisation

Correct

1000	02	04	06	08	10	12	14	16	18
10	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9

★ int a[10];
But, a[10] = {10, 11. - - - .19}; // wrong
∵ a[10] points only to 10th element. ✗

★ int a[] = {10, 11, 12. - - .19}
No. of elements not shown // Correct ✓

★ int a[]; // wrong (undefined size) (don't know how many bytes to create) ✗

★ int a[5] = {10, 11, 12, 13, 14, 15, 16. - - -19}; // Correct ✓
array size is 5 (10 bytes m/m available)
for these 10 bytes No m/m available

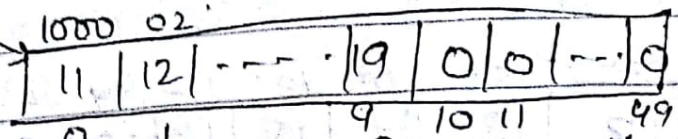
10	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9

only mine m/m area
not mine m/m area (You can't keep out of ur m/m)

· You are loosing data.
Now, printf(a[6]) = garbage.
∵ this m/m doesn't belong to us.

Select
Date : 25/5/13
Page no. 168

☆ int a[50] = {11, 12, ----- 19}; 1000 bytes m/m allocated



By default zero will be stored in remaining m/m area

☆ Array is also called Static data structure ∵ at the starting only you fix the size.

→ Array "a" means 1st byte address
⇒ a = 1000
⇒ *a = 10 // 2 bytes data will be Extracted (from 1000 onwards 2 byte data extracted).
value at a present at 1000 & 1001.

purpose of Data Type ⇒ at the time of storing how many bytes to be stored & how many bytes to retrieve at retrieval time.

Declarative
All the programming languages are declarative (before usage of variable declare it first).

Date: 25/5/13
Page no. 169

✓ a = 1000
a+0 = 1000
*(a+0) = 10 (value a+1000)

a[0] = *(a+0)

✓ a = 1000
a+1 = 1002 a[1] = *(a+1) = 11 (value at 1002)
↳ skipping 1 element i.e. 2 bytes ∵ it is integer array

✓ float a[10];
a = 1000
a+1 = 1004 // 4 bytes skipping

✓ char *a[10]; // "array of pointers"
In all the locations we are storing addresses.

✓ a = 1000

a+5 = a[5] 1010
skipping 5 element

*(a+5) = *(1010) = 15 = a[5]

a = Base address = 1000
a+9 = 1018
skipping 9 elements a[9] = *(a+9) = 19

Select
Date: 25/8/13
Page no. 170

for user → for compiler → Commutative property

Note: a[i] = *(a+i) = *(i+a)
(easy to write) skipping i elements

Note: 1

*(a+i) = *(i+a)
a[i] = i[a] = *(a+i) = *(i+a)

1D array

(due to commutative property)

Note 2: a[i][j] = b[j] = *(b+j)

b

2D array

↓

*(a[i]+j)

↓

((a+i)+j)

↓

base address

i = row
j = column

→ Collection of 1D arrays is called 2D array.

→ 1D array is collection of elements.

→ 3D array: collection of 2D arrays

Select
Date: 25/5/13
Page no. 171

1-D array

✓ a[0....100] - No. of elements in 'a'
⇒ 100 − 0 + 1 = 101
(last ele, 1st ele)

c = 2 (Size of each element)

✓ a[7, 8, 9, 10] No. of elements in 'a' ⇒ 10 − 7 + 1 = 4

✓ a[50....150] No. of elements in a ⇒ 150 − 50 + 1 ⇒ 101 elements

a[lb.... ub]
# of elements in array a = ub − lb + 1

✓ 0 1 2 3 4 5 6
(0 1 2 3 before, 4 including this one)

4 − 0 + 1 = 5
(4 − 0: before 4th element, how many elements; 1: including 4th element)

✓ 10 11 12 13 14 15 16

⇒ 14 − 10 + 1 = 5 elements

✓ a[0.....100]

100 − 0 + 1 = 101 [including 100 before no. of elements]

100 − 0 = 100 [before 100 no. of elements]

Loc (a[25]) = ~~(25-0)+1~~

address

$1000 + (25-0) \times 2$

from 1000, skip 25 elements, each ele size

$= 1050$

Loc (a[98]) = $1000 + (98-0) \times 2$

address, before 98 how many

$= 1000 + 196$

$= 1196$

~~Loc (a[55....71])~~

Eg a[55.....71]

$71-55+1=17$

B.A = 999, C = 5 bytes

$Loc(a[64]) = 999 + (64-55) \times 5$

$= 999 + 45$

$= 1044$

$(64-55) \times 5$

9×5

45

Ex a[-50.....+50]

101 elements

B.A = 0, C = 20 bytes

$Lo((a[0]) = 0 + (0-(-50)) \times 20$

$= ~~0+250~~$

$= ~~250~~$

$= 1000$

Date: 25/5/13
Page no. 173

Note: FORMULA

a[lb ---- ub]
B.A = Base address
C = Size of element

$$\text{Loc}(a[iJ]) = B.A. + [(i - lb)] \times C$$

(Loc(a[iJ]) = i^th element address)

Q Why should array start from 0?

Case a[0.---.100]
B.A. = 5000, C = 10

$$\text{Loc}(a[55]) = 500 + (55 - 0) * 10$$
$$= 1050$$

(55 → offset; not req.)

(Beneficial no need of subtraction or offset value calculation)

Case a[1.---.100]
B.A. = 500, C = 10

$$\text{Loc}(a[55]) = 500 + (55 - 1) * 10$$
$$= 1040$$

(55 - 1 → offset)

(offset calculation is necessary)

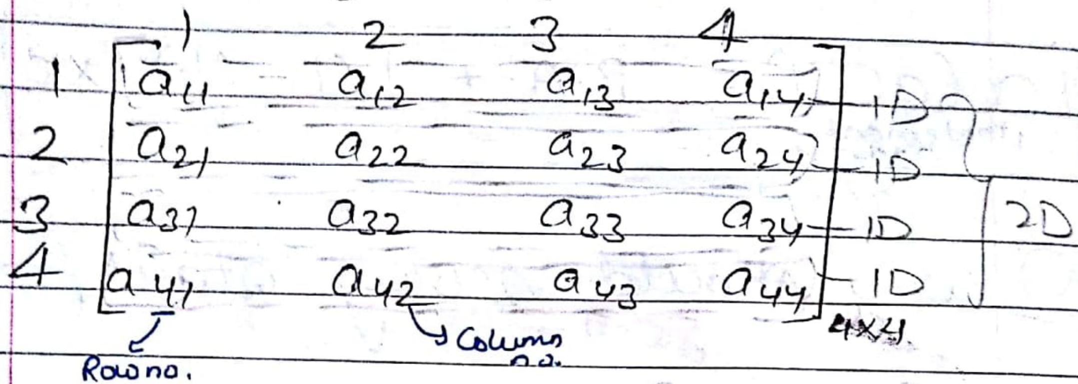
(Extra subtraction or offset value calculation)

Note If array starts from '1' then offset calculation is required (takes extra time).

2-D array

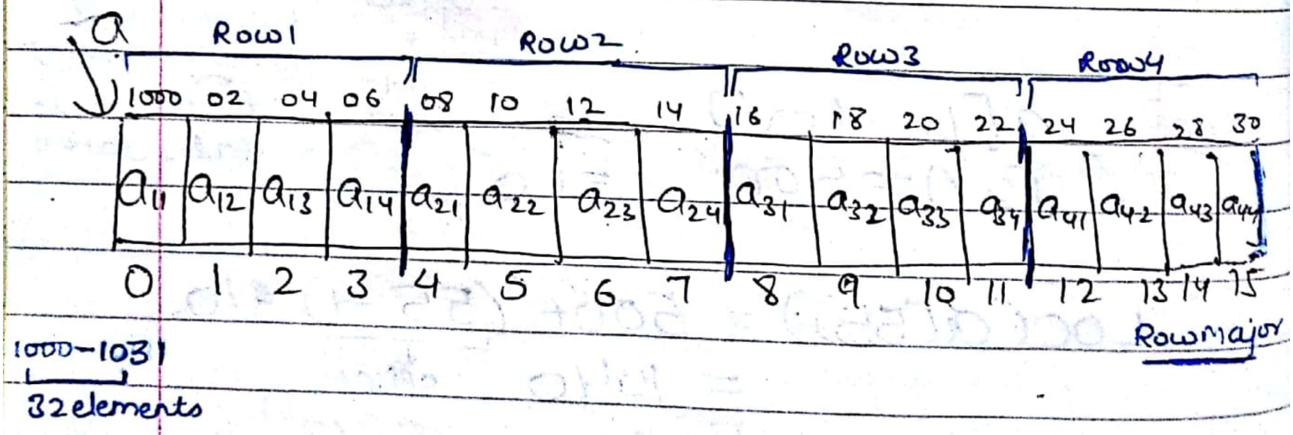
int a[4][4] = 32 bytes (4x4x2)

(Rows → first [4], Columns → second [4])



int a[4][4] = int a[1....4, 1....4]

(int → 2 bytes; 1st Row no., Last Row no., 1st Column no., Last Column no.; 4-1+1, 4-1+1)



- 2D array is collection of 1D arrays.
- Row by Row storing in M/m = Row Major
- Column by Column storing in M/m = Column Major

or
Collection of elements

Select
Date 25/5/13
Page no. 175

* Greatest property of array = Random access

LOC (a[4][3]) = B.A. (1000) + [(4-1) * 4 + (3-1)] × 2 = 1028

4th row, 3rd column

(4-1): cross 3 rows; 4: every row has 4 elements; (3-1): then crossing 2 columns

- Firstly goto 4th row then by accessing 3rd element

4th row = cross 3 rows

LOC (a[1][4]) = 1000 + [(1-1) * 4 + (4-1)] * 2

(1-1): before 1st row how many rows are there? (Row)
4: how many elements / columns in a row
(4-1): column, after going to the row in which column we want to go

= 1000 + 6 = 1006

Example 2

a[55 .... 99, 71 .... 125]

lb1 = 55, ub1 = 99, lb2 = 71, ub2 = 125

99 - 55 + 1 = 45 rows; 125 - 71 + 1 = 55 columns.

B.A. = 0, c = 2, Row Major order

LOC(a[88][92]) = 0 + [(88-55) * 55 + (92-71)] * 2

= [33 × 55 + 21] × 2

= [1815 + 21] * 2

= [1836] * 2 = 3672

Example3

a[-5 ... +5, 5 ... 55]

↓ 5-(-5)+1 = 11 rows

↓ 55-5+1 = 51 columns

RMO, B.A = 1000, C = 10

Loc(a[-1][50]) = 1000 + [(-1-(-5))*51 + (50-5)]*10

= 1000 + [204 + 45]*10

= 1000 + 2490

= 3490

Note:

Formula

$n_r = ub1 - lb1 + 1$, $n_c = ub2 - lb2 + 1$

a[lb1 ... ub1, lb2 ... ub2]

B.A = Base address

C = Size of element

Row major order

RMO

Loc(a[i][j]) = B.A + [(i - lb1) * (ub2 - lb2 + 1) + (j - lb2)] * C

Formula

$n_r = ub1 - lb1 + 1$, $n_c = ub2 - lb2 + 1$

a[lb1 ... ub1, lb2 ... ub2]

B.A = Base address

C = Size of element

Column Major order

CMO

Loc[a[i][j]] = B.A + [(j - lb2) * n_r + (i - lb1)] * C

(n_r = ub1 - lb1 + 1)

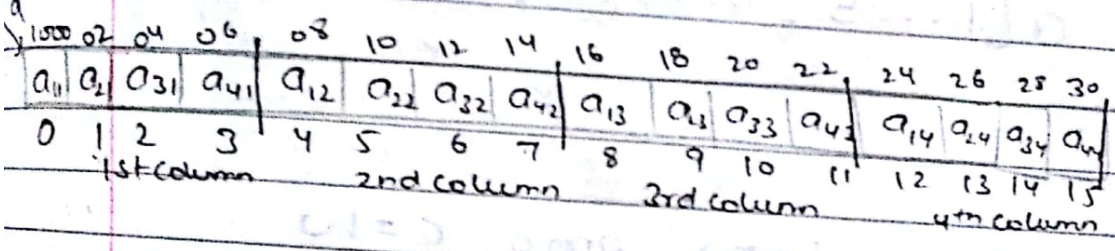
Select
Date : 25/5/13
Page no. 177

Column Major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

column by column store

int a[4][4]



$$LOC[a[3][4]] = 1000 + [(4-1)*4 + (3-1)]*2$$
$$= 1000 + [12 + 2]*2$$
$$= 1028$$

$$LOC[a[2][3]] = 1000 + [(3-1)*4 + (2-1)]*2$$
$$= 1000 + [8+1]*2$$
$$= 1018$$

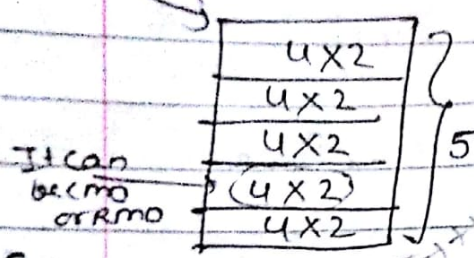
$$LOC[a[3][1]] = 1000 + [(1-1)*4 + (3-1)]*2$$
$$= 1000 + 4 = 1004$$

$$LOC[a[2][2]] = 1000 + [(2-1)*4 + (4-1)]*2$$
$$= 1014$$

3-D arrays = collection of 2D arrays

int a[5][4][2] ⇒ five 2-D arrays each of size [4][2].

eg



Ex

a[1 .... 5 , 1 .... 4 , 1 .... 2]

- 5−1+1 = 5 ⇒ No. of 2D arrays
- 4−1+1 = 4 ⇒ # Rows in each 2D array
- 2−1+1 = 2 ⇒ # Columns in each 2D array

$n_r \times n_c$ = Array size

B.A. = 1000, RMO, C = 10

LOC[a[4][2][1]] = 1000 + [(4−1) × 8 (crossing 3 2D arrays; 8 = each array size) + {(2−1) * 2 + (1−1)] * 10

= 1000 + [24 + 2] * 10

= 1260

Note Formula

a[lb1 ... ub1, lb2 ... ub2, lb3 ... ub3]

BA, C, (RMO), n_a, n_r, n_c

asize = $n_r \times n_c$ (array size)

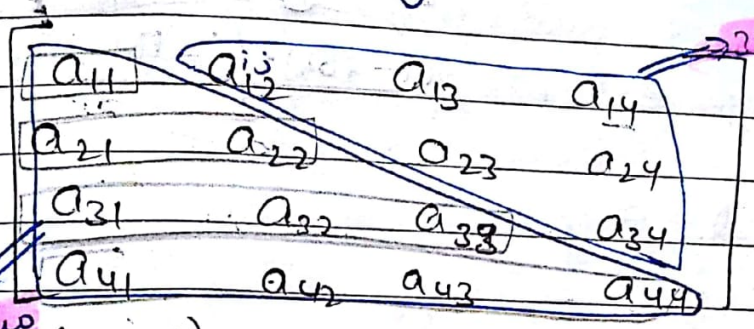
LOC[a[i][j][k]] = BA + [(i − lb1) × asize + (j − lb2) × n_c + (k − lb3)] * C

Select
Date: 25/5/13
Page no. 179

CMO

$$Loc[a[i][j][k]] = BA + [(i - lb1) * asize + (k - lb3) * n_r + (j - lb2)] * c$$

Lower Triangular Matrix



$$n^2 - \frac{n(n+1)}{2}$$

(total elements) (upper part)

RMO

If $j \geq i$ (upper part) need to be zero

If $i \geq j$ (lower part) need to be stored

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}	a_{41}	a_{42}	a_{43}	a_{44}
0	1	2	3	4	5	6	7	8	9

$$Loc[a[4][3]] = 1000 + [(4-1)NNS + (3-1)] * 2$$
$$= 1000 + [6+2] * 2$$
$$= 1016$$

$$Loc[a[2][1]] = 1000 + [(2-1)NNS + (1-1)] * 2$$
$$= 1000 + 2 = 1002$$

Select
Date : 25/5/13
Page no. 180

Formula

Note: a[lb1 ... ub1, lb2 ... ub2]
BA, C, LTM (Lower Triangular Matrix)
RMO.

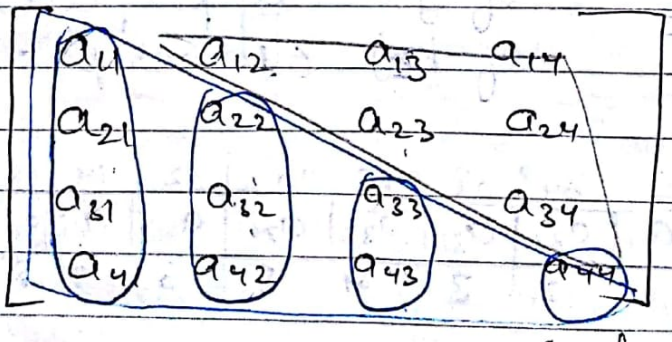
$$Loc[a[i][j]] = BA + C * \left[\underbrace{(i - lb1) \text{ NNS}}_{\text{sum of } (i-lb1) \text{ natural no.}} + (j - lb2) \right]$$

$$\text{sum of } (i-lb1) \text{ natural no.} = \frac{(i-lb1)\cdot(i-lb1+1)}{2}$$

CMO

a

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{21}	a_{31}	a_{41}	a_{22}	a_{32}	a_{42}	a_{33}	a_{43}	a_{44}
0	1	2	3	4	5	6	7	8	9



← from this side sum of n natural no, formula will work

$$Loc[a[4][3]] = 1000 + \left[\underbrace{\frac{(4*5)}{2}}_{\text{Cross all columns}} - \underbrace{\frac{2*3}{2}}_{\text{subtract Extra column crossed}} + (4-3) \right] * 2$$

(2 Extra column we have to cross)

$$= 1000 + [10 - 3 + 1] * 2$$
$$= 1000 + 8 * 2$$
$$= 1016$$

Ex 1 2 3 (4 5 6 7 8)

$\frac{8 \times 9}{2} - \frac{5 \times 6}{2} = 36 - 15 = 21$

(cross all) (come back again to 4)

Example 2

LOC [a[2][1]] =

$1000 + \left[\frac{4 \times 5}{2} - \frac{4 \times 5}{2} + (2-1)\right] \times 2$ ← 4 extra columns crossed.

$= 1000 + [10 - 10 + 1] \times 2$

$= 1000 + 2 = 1002$

Ex a[0 ... 99, 0 ... 99] (=100, =100)

BA = 1000, c = 10, LTM, CMO

LOC [a[50][45]]

$= 1000 + \left[\frac{100 \times 101}{2} - \frac{(99-45+1)(99-45+2)}{2} + (50-45)\right] \times 10$

(cross all columns.)

(99 columns) $= 36150$

0, 1, 2 ... 44 45 46 ... 99

Note: Formula

a[lb1 ... ub1, lb2 ... ub2] $n_r = ub1 - lb1 + 1$

BA, c, LTM, CMO $n_c = ub2 - lb2 + 1$

LOC[a[i][j]] = $BA + \left[\frac{n_c * (n_c+1)}{2} - (ub2-j+1)(ub2-j+2) + (i-j)\right] * c$

(crossing all columns) (1 am at ub2 column) (crossing unnecessary columns) (I am at j^th column, j^th row)

Three dimensional Array

int a[2] [3] [4] — 2 matrices of size 3X4 or 2 arrays of size 3X4

- [2] ⇓ 1…2 (Array)
- [3] ⇓ 1…3 (Row)
- [4] ⇓ 1…4 (Column)

1 — [3X4] → 12 elements of 24 bytes
2 — [3X4] → 12 elements of 24 bytes
⇒ 48 bytes

Question

B.A. = 1000, RMO, C = 10

$$LOC(a_{2,3,4}) = 1000 + [(2-1) * 12 + \{(3-1) * 4 + (4-1)\}] * 10$$

(2−1): 1 array crossed covering all rows & columns in that array; 12 = 3x4

1 [3X4] ⇒ 12
2 [3X4] ⇒ 12
1 array has 12 elements

= 1230

Question

int a[−5 …. +5, −10 …. +50, −25 …. +25]
(lb1 = −5, ub1 = +5, lb2 = −10, ub2 = +50, lb3 = −25, ub3 = +25)

B.A. = 0, CMO, c = 5

$n_a = 5-(-5)+1 = 11$

$n_r = 50-(-10)+1 = 61$

$n_c = 25-(-25)+1 = 51$

a.size = $n_r \times n_c$ = 61 * 51

$$LOC(a_{0,0,0}) = 0 + [(0-(-5)) \times 61 \times 51 + [(0+25) * 61 + (0+10)]] * 5$$

(0−(−5)): 5 arrays crossed

= 85450

In General,

array row column

A[lb1 ... ub1, lb2 ... ub2, lb3 ... ub3]

B A, c, RMO, c, nr, nc, na

$$LOC(a_{i,j,k}) = B.A. + [(i-lb1)*n_r*n_c + (j-lb2)*n_c + (k-lb3)]*C$$

CMO

$$LOC(a_{i,j,k}) = B.A. + C*[(i-lb1)*n_r*n_c + (k-lb3)*n_r + (j-lb2)]$$

---

optional

int a[5] = {10, 20, 30, 40, 50}

or

a[]

int a[2][3] = {10, 20, 30 ... 60};

int a[5] = { }; // stores 5 0's by default

int a[5]; // stores garbage value

int a[4] = {10, 20, 30, 40, 50}; // 5th element is lost

int a[]; // Error

int a[] = { }; // Error

int a[][3] = {10, 20, 30 . . . 60};

optional mandatory

int a[2][] = {10, 20. . . 60}; // Error?

this is mandatory

int a[1][2][3] = {1, 2, 3, 4, 5, 6}; // Correct

int a[][2][3] = {1, 2, 3, 4, 5, 6}; // Correct

optional mandatory mandatory

int a[][][] = {1, 2, 3, 4, 5, 6} // Error

Always LHS is optional & RHS is always mandatory.

Note: In the Multidimensional (m)-array (m-1) people are mandatory.

Question Main() What is the O/P?

```
{
    int a[3][3] = {10, 20, 30, 40, 50, 60, 70, 80, 90};
    printf("%d", ((a==1000 * a) && (*a == a[0])));
}
```

(a==1000 * a) → 1000 * 1000 → 1

(*a == a[0]) → 10 * 10 → 1

O/P: 1 (1 && 1 = 1)

a →	0	1	2
1000 → 0	1000 10	1002 20	1004 30
1006 → 1	1006 40	1008 50	1010 60
1012 → 2	1012 70	1014 80	1016 90

a+1 = 1006

↳ means 1 rows skipped (6 Bytes of 3 elements)

printf("%d", ((a==*a) && (*a==a[0])); ⇒ O/P = 0

(a: 1000, *a: 10 → Not Equal → 0) && (*a: 1000, a[0]: 1000 → 1) = 0

In 2D array, 2 stars are required to get the value.

b[i][j] =

b[j]

⇓

*(b+j)

⇓

*(a[i]+j)

⇓

((a+i)+j)

(Select Row, Select column)

Row not selected	a = 1000, a+0 = 1000 a+1 = 1006 (1 row skipped), a+2 = 1012 (2 rows skipped)

(Without 1 star row selection is not over)

only Row is selected	*(a+0) = *1000 = 1000 *(a+1) = *1006 = 1006 *(a+2) = *1012 = 1012

(1 row is selected but we don't know which column)

C.P

$a+1 = 1006$

$(a+1)+1 = 1012$

$*(a+1) = 1006$

$*(a+1)+1 = 1008$ (row selected, skip 1 column but column not selected)

$(a+2) = 1012$

$(a+2)+1 = 1018$ (1 more row skipped)

$*(a+2) = 1012$ (2 rows skipped, 3rd row selected)

$*(*(a+1)+1) = 50 = a[1][1]$

(1000 → 1006 (1 row skipped) → 1006 → 1008 (column skip, column not selected) → *1008 (inside column))

$*(*(a+1)+1)+1 = 50+1 = 51 = a[1][1]+1$

(1000 → 1006 → 1006 → 1008 → *1008 = 50 → 50+1 = 51)

Question: What is the output?

```
Main ()
{
int a[2][3][2] = {2,4,7,8,3,4,2,2,2,3,3,4};
printf("%u %u %u %d", a, *a, **a, ***a);
printf("%u %u %u %d", a+1, *a+1, **a+1, ***a+1);
printf("%u %d %u %d", *(*(a+2)+3), *(*a[1]+2),
        *(a+2), *(*(a[1]+1)));
```

(a = 1000, *a = 1000, **a = 1000, ***a = 2)

(a+1 = 1012 (array skip), *a+1 = 1004 (row skip), **a+1 = 1002 (column skip), ***a+1 = 3)

(*(*(a+2)+3): 1000 → 1024 → 1036; *(*a[1]+2): 1012 → 1012 → 1016 (column skip) → 2)

(*(a+2): 1000 → 1024 → 1024; *(*(a[1]+1)): 1012 → 1016 (1 row skip) → 1016 → 2)

printf("%u %u %u %u %d", a+1, a[1]+1, a[1][2]+1, a[1][2][3]);

1012 1016 1 column skip

1 row skip (4 bytes) 1020 1022 Garbage + Garb

O/P : 1000, 1000, 1000, 2
1012, 1004, 1002, 3
1036, 2, 1024, 2
1012, 1016, 1022, Garbage.

a	0	1
1000	1000 2	02 4
	04 7	06 8
	08 3	10 4
1012	12 2	14 2
	16 2	18 3
	20 3	22 4

1000 → a[0][0] 0
1004 → a[0][1] 1
1008 → a[0][2] 2
1012 → a[1][0] 0
1016 → a[1][1] 1
1020 → a[1][2] 2

Array = 12 bytes
Row = 4 bytes
Column = 2 bytes

unary operator (*, ++) have higher priority then Binary operator.

a = 1000

*a = 1000 ≡ *(a+0) = *1000 = 1000

**a = 1000 (1 array & 0 row selected)

a+1 = 1012 (1 array skipped) (we are on 2nd array)

*(a+1) = 1012

**(a+1) = 1012

***(a+1) = 2

a = 1000

*a = 1000

**a = 1000

***a = 1

a+2 = 1024 (2 arrays skipped. each array is of 12 bytes ∴ of 6 elements in an array)

*a+1 = 1004

array : 1 row skipped

**a+1 = 1002

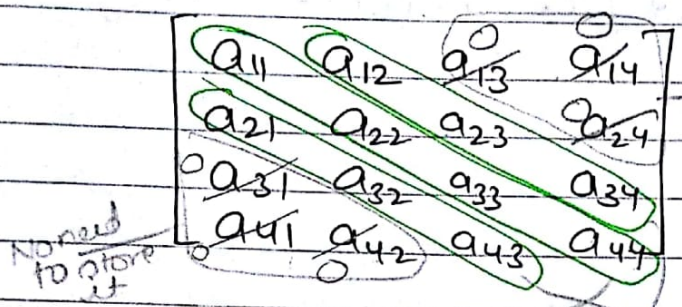
array & row selected, 1 column skipped

***a+1 = 3

value selected

ARRAYS

Tridiagonal Matrix



No. of elements in Tridiagonal Matrix = N + (N-1) + (N-1) = 3N - 2

N → Principal diagonal. (N-1) → upper diagonal (N-1) → lower diagonal

a) Row Major Order

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{12}	a_{21}	a_{22}	a_{23}	a_{32}	a_{33}	a_{34}	a_{43}	a_{44}
1st Row		2nd row			3rd row			4th row	

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{bmatrix}$$

$$LOC(a_{i,j}) = 1000 + 2 * [3*(3-1) - 1 + (4-3+1)]$$

(2 rows containing 3 elements; from 1st row skip 2 elements; 3rd row of element reached; 2 columns skipped)

$$= 1014$$

$$LOC(a_{i,j}) = 1000 + 2 * [3*(4-1) - 1 + (4-4+1)]$$

(Every Row contain 3 elements)

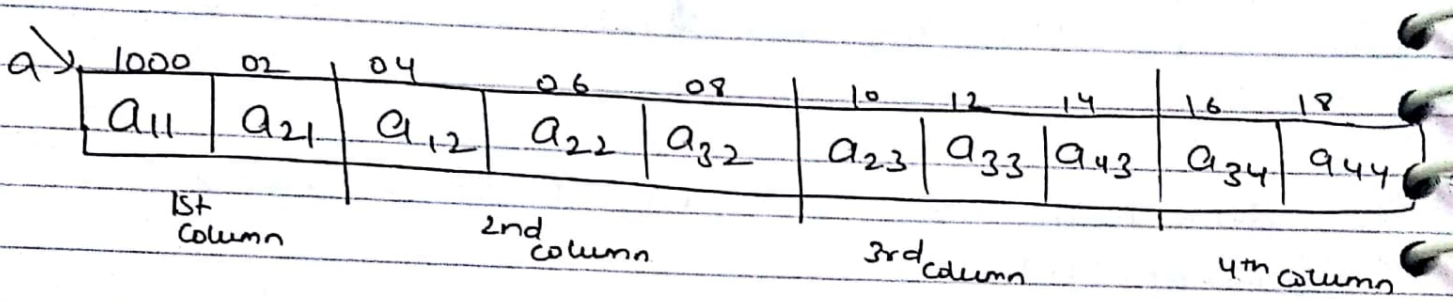
↳ But out of all above 3 rows, 1 row contain only 2 elements so subtract that by 1.

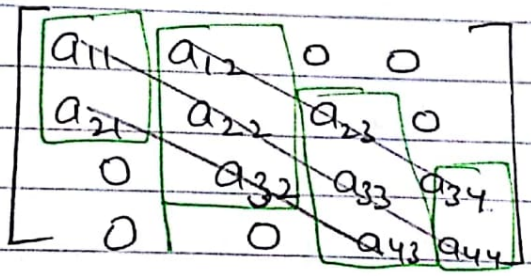
$$= 1018$$

general formula :-)

$$Loc(a_{i,j}) = B.A. + c * [3 * (i - lb1) - 1 + (j - i + 1)]$$

b) Column Major





$$Loc(a_{3,4}) = 1000 + 2 * [3(4-1) - 1 + (3-4+1)]$$

i j

3 columns skipped, 2 from 1st column, 1 element subtracted

Row of the element.

$$= 1016$$

$$Loc(a_{3,2}) = 1000 + 2 * [3 \cdot (2-1) - 1 + (3-2+1)]$$

i j

1 column skipped

2 rows skipped

$$= 1008$$

general formula :-)

$$Loc(a_{i,j}) = B.A. + c * [3 * (j - lb2) - 1 + ((i-j)+1)]$$

(C) Storing diagonal by diagonal (Principle, lower, upper)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Store 1st principle diagonal then lower diagonal after that upper diagonal.

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{22}	a_{33}	a_{44}	a_{21}	a_{32}	a_{43}	a_{12}	a_{23}	a_{34}
Principle				lower			upper		

Loc $(a_{i,j})$ =>

if $(i==j)$ Principle

$$Loc(a_{i,j}) = B.A. + (i - lb1) * C$$

if $(i>j)$ lower

$$Loc(a_{i,j}) = B.A. + (\underbrace{N}_{\text{from the principle diagonal having N elements}} + j - 1) * C \quad [\because \text{it has N elements of N}\times\text{N matrix}]$$

if $(i<j)$ upper

$$Loc(a_{i,j}) = BA + (\underbrace{N + N-1}_{\text{from principal \& lower diagonal}} + i - 1) * C$$

Z-Matrix

lbl

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

ubl

$N + N + N - 2 = 3N - 2$

- N: 1st Row
- N: last Row
- N-2: 1 diagonal in which 2 elements are removed.

Storing in Z-format

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{12}	a_{13}	a_{14}	a_{41}	a_{42}	a_{43}	a_{44}	a_{23}	a_{32}

1st Row | Last Row | Remaining elements in diagonal

$LOC(a_{i,j})$

if $(i == lbl)$ 1st Row

$$LOC(a_{i,j}) = BA + (j - lb2) * C$$

if $(i == ubl)$ Last Row

$$LOC(a_{i,j}) = B.A. + (N + (j - lb2)) * C$$

(N: 1st Row is over)

if $(i \neq lbl \;\&\&\; i \neq ubl \;\&\&\; (i+j) == (N+1))$ diagonal

$$Loc(a_{i,j}) = BA + [N + N + (i-2)] * C$$

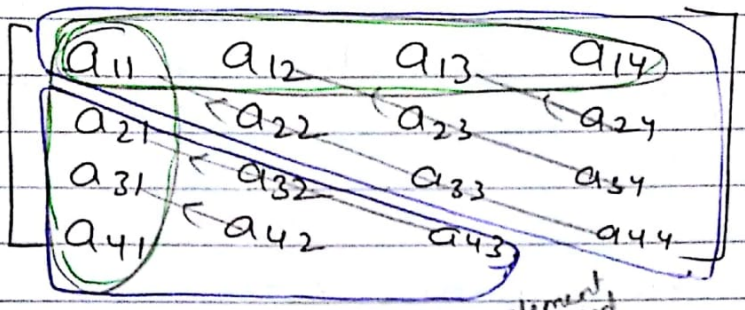
(N: 1st Row over; N: last Row over)

Toeplitz Matrix

Condition

if $A[i,j] = A[i-1, j-1]$

$\forall i,j$ where $i>1$ && $j>1$

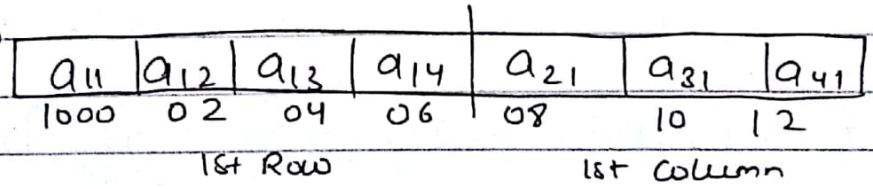


for a44 return a11
for a33 " a11
for a43 return a21
for a42 return a31

$\Rightarrow \underbrace{N}_{1 \text{ row}} + \underbrace{N-1}_{1 \text{ column}}$ ← element included in Row

1st Row is covering upper matrix
1st column is covering lower matrix

a) Store 1st Row then 1st column



Loc $(a_{i,j})$ = if $(i \leq j)$ (upper)

Loc $(a_{i,j}) = BA + (j-i) * C$

Loc $(a_{i,j})$ = if $(i \geq j)$ (lower)

Loc $(a_{i,j}) = B \cdot A \cdot + C * (N + (i-j) - 1)$

Three dimensional Array

int a [2] [3] [4]

- [2] ⇒ 1…2 (Array)
- [3] ⇒ 1…3 (Row)
- [4] ⇒ 1…4 (Column)

2 matrices of size 3X4
or
2 arrays of size 3X4

1 — [3X4] → 12 elements of 24 bytes
2 — [3X4] → 12 elements of 24 bytes
⇒ 48 bytes

Question

B.A. = 1000, RMO, C = 10

$$LOC(a_{2,3,4}) = 1000 + [(2-1) * 12 + [(3-1) * 4 + (4-1)]]$$

(2−1): 1 array crossed; 12 (3x4): crossing all rows & columns in that array

1 [3X4 = 12]
2 [3X4 = 12]
1 array has 12 elements

= 1230

Question

int a [−5 …. +5, −10 …. +50, −25 …. +25]
(lb1 = −5, ub1 = +5, lb2 = −10, ub2 = +50, lb3 = −25, ub3 = +25)

B.A. = 0, CMO, C = 5

$n_a = 5-(-5)+1 = 11$

$n_r = 50-(-10)+1 = 61$

$n_c = 25-(-25)+1 = 51$

a.size = $n_r \times n_c = 61 * 51$

$$LOC(a_{0,0,0}) = 0 + [(0-(-5)) \times 61 \times 51 + [(0+25) * 61 + (0+10)]] * 5$$

(0−(−5)): arrays crossed

= 85450

In General,

array row column

A[lb1 ... ub1, lb2 ... ub2, lb3 ... ub3]

B.A, c, RMO, c, nr, nc, na

$$Loc(a_{i,j,k}) = B.A. + [(i - lb1) * n_r * n_c + (j - lb2) * n_c + (k - lb3)] * c$$

CMO

$$Loc(a_{i,j,k}) = B.A. + c * [(i - lb1) * n_r * n_c + (k - lb3) * n_r + (j - lb2)]$$

optional

int a[5] = {10, 20, 30, 40, 50}

or

a[]

int a[2][3] = {10, 20, 30 ... 60};

int a[5] = { }; // stores 5 0's by default

int a[5]; // stores garbage value

int a[4] = {10, 20, 30, 40, 50}; // 5th element is ignored

int a[]; // Error

int a[] = { }; // Error

int a[][3] = {10, 20, 30 . . . 60};

optional, mandatory

int a[2][] = {10, 20 . . . 60}; // Error?

this is mandatory

int a[1][2][3] = {1, 2, 3, 4, 5, 6}; // correct

int a[][2][3] = {1, 2, 3, 4, 5, 6}; // correct

optional, mandatory, mandatory

int a[][][] = {1, 2, 3, 4, 5, 6} // Error

Always LHS is optional & RHS is always mandatory.

Note: In the Multidimensional (m)-array (m-1) people are mandatory.

Question Main() What is the O/P?

```
{
    int a[3][3] = {10, 20, 30, 40, 50, 60, 70, 80, 90};
    printf("%d", ((a==a[0]) && (*a==a[0])));
}
```

(a==a[0]) → 1000 == 1000 → 1; (*a==a[0]) → 10 == 10 → 1

O/P: 1 (1 && 1 = 1)