

`int` → 2 byte
`float` → 4 byte
`char` → 1 byte.

Array → (A) . CLASSMATE
Date _____
Page _____

Module II

01 = ARRAYS

- * It is a ~~set~~ data str containing a no. of data values (same type).
- * Data str → format for organizing & storing data.
→ no meaningful values.
- * * Data str is represented with the help of characters like A-Z, a-z, 0-9 or spec char.
- * * Info is organized / classified data which has some meaningful values for the receiver.
eg → what is yr name ?, meaning.
eg for data ⇒ mean xoy si tahu
(ng to understand)
- * Array containing 6 data values.

10	60	7	23	70	43	✓
'P'	5	5.5	X			[Static memory allocation]

- * Declaration of (A) →
eg → `int arr[5];`
syntax → datatype array name [no of elements].

arr [1 2 3 4 5], Total size of memory =
5 * size of (int);
5 * 2 = 10
(int)

- * Initialization of (A) →

a) Compile time initialization =

`arr[5] = { 1, 2, 3, 4, 5 };`

(Qn)

`arr[5] = { 1, 2, 3, 4, 5 };`

`for (int i = 0; i < 5; i++)`

D) Run time initialization =

```
int arr[5], i;
for (i = 0; i < 5; i++)
{
    scanf ("%d", &arr[i]);
}
```

(1 by one
execute)

* Accessing elements =

`arr [10 20 30]`

eg → arr[0];

using index no.

⇒ Array operations = (i) Traversal

(a) Traversal

1. start

2. using for loop from 0 to n

n → size of array

3. Access every element of array by

a[index] , (a[i])

4. print the elements.

`printf ("%d", a[i]);`

```
for (i = 0; i < size; i++)
{
    scanf ("%d", &a[i]);
}
```

```
}
```

b) At end =

```
printf ("inserted value");
scanf ("%d", &dn);
```

```
if (dn > limit)
    limit = limit + 1;
```

```
int a[20], size;
printf ("Enter size");
scanf ("%d", &size);
printf ("Enter elements");
for (i = 0; i < size; i++)
{
    printf ("%d", a[i]);
}
return 0;
```

main for traversal.

(2) Insertion =

a) At beginning =

after →

```
printf ("Inserted value");
scanf ("%d", &dn);
for (i = limit - 1; i >= 0; i--)
{
    a[i + 1] = a[i];
}
a[0] = dn;
```

`a[0] = dn;`

`limit = limit + 1;`

c) At specific point =

```
printf("Enter the position:");
scanf("%d", &position);
printf("Selected value : ");
scanf("%d", &n);
for(i=limit-1; i>=pos; i--) {
    a[i+1] = a[i];
}
```

```
i = i+1;
j = position;
limit = limit+1;
```

③ deletion =

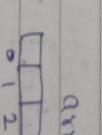
a) At beginning =

```
for(i=0; i<n-1; i++) {
    a[i] = a[i+1];
};
```

b) At specific point =

```
for(i=position; i<n-1; i++) {
```

* Accessing array name [index].

* Initialization =  arr[0], arr[1]

arr[5] = {1, 2, 3, 4, 5};

arr[] = {1, 2, 3, 4, 5};

* Initialization =

arr[5];

arr[0] = 1;

arr[1] = 2;

arr[2] = 3;

arr[3] = 4;

arr[4] = 5;

arr[5] = 6;

arr[6] = 7;

arr[7] = 8;

arr[8] = 9;

arr[9] = 10;

arr[10] = 11;

arr[11] = 12;

arr[12] = 13;

arr[13] = 14;

arr[14] = 15;

arr[15] = 16;

arr[16] = 17;

arr[17] = 18;

arr[18] = 19;

arr[19] = 20;

arr[20] = 21;

arr[21] = 22;

arr[22] = 23;

arr[23] = 24;

arr[24] = 25;

arr[25] = 26;

arr[26] = 27;

arr[27] = 28;

arr[28] = 29;

arr[29] = 30;

arr[30] = 31;

arr[31] = 32;

arr[32] = 33;

arr[33] = 34;

arr[34] = 35;

arr[35] = 36;

arr[36] = 37;

arr[37] = 38;

arr[38] = 39;

arr[39] = 40;

arr[40] = 41;

arr[41] = 42;

arr[42] = 43;

arr[43] = 44;

arr[44] = 45;

arr[45] = 46;

arr[46] = 47;

arr[47] = 48;

arr[48] = 49;

arr[49] = 50;

arr[50] = 51;

arr[51] = 52;

arr[52] = 53;

arr[53] = 54;

arr[54] = 55;

arr[55] = 56;

arr[56] = 57;

arr[57] = 58;

arr[58] = 59;

arr[59] = 60;

arr[60] = 61;

arr[61] = 62;

arr[62] = 63;

arr[63] = 64;

arr[64] = 65;

arr[65] = 66;

arr[66] = 67;

arr[67] = 68;

arr[68] = 69;

arr[69] = 70;

arr[70] = 71;

arr[71] = 72;

arr[72] = 73;

arr[73] = 74;

arr[74] = 75;

arr[75] = 76;

arr[76] = 77;

arr[77] = 78;

arr[78] = 79;

arr[79] = 80;

arr[80] = 81;

arr[81] = 82;

arr[82] = 83;

arr[83] = 84;

arr[84] = 85;

arr[85] = 86;

arr[86] = 87;

arr[87] = 88;

arr[88] = 89;

arr[89] = 90;

arr[90] = 91;

arr[91] = 92;

arr[92] = 93;

arr[93] = 94;

arr[94] = 95;

arr[95] = 96;

arr[96] = 97;

arr[97] = 98;

arr[98] = 99;

arr[99] = 100;

arr[100] = 101;

arr[101] = 102;

arr[102] = 103;

arr[103] = 104;

arr[104] = 105;

arr[105] = 106;

arr[106] = 107;

arr[107] = 108;

arr[108] = 109;

arr[109] = 110;

arr[110] = 111;

arr[111] = 112;

arr[112] = 113;

arr[113] = 114;

arr[114] = 115;

arr[115] = 116;

arr[116] = 117;

arr[117] = 118;

arr[118] = 119;

arr[119] = 120;

arr[120] = 121;

arr[121] = 122;

arr[122] = 123;

arr[123] = 124;

arr[124] = 125;

arr[125] = 126;

arr[126] = 127;

arr[127] = 128;

arr[128] = 129;

arr[129] = 130;

arr[130] = 131;

arr[131] = 132;

arr[132] = 133;

arr[133] = 134;

arr[134] = 135;

arr[135] = 136;

arr[136] = 137;

arr[137] = 138;

arr[138] = 139;

arr[139] = 140;

arr[140] = 141;

arr[141] = 142;

arr[142] = 143;

arr[143] = 144;

arr[144] = 145;

arr[145] = 146;

arr[146] = 147;

arr[147] = 148;

arr[148] = 149;

arr[149] = 150;

arr[150] = 151;

arr[151] = 152;

arr[152] = 153;

arr[153] = 154;

arr[154] = 155;

arr[155] = 156;

arr[156] = 157;

arr[157] = 158;

arr[158] = 159;

arr[159] = 160;

arr[160] = 161;

arr[161] = 162;

arr[162] = 163;

arr[163] = 164;

arr[164] = 165;

arr[165] = 166;

arr[166] = 167;

arr[167] = 168;

arr[168] = 169;

arr[169] = 170;

arr[170] = 171;

arr[171] = 172;

arr[172] = 173;

arr[173] = 174;

arr[174] = 175;

arr[175] = 176;

arr[176] = 177;

arr[177] = 178;

arr[178] = 179;

arr[179] = 180;

arr[180] = 181;

arr[181] = 182;

arr[182] = 183;

arr[183] = 184;

arr[184] = 185;

arr[185] = 186;

arr[186] = 187;

arr[187] = 188;

arr[188] = 189;

arr[189] = 190;

arr[190] = 191;

arr[191] = 192;

arr[192] = 193;

arr[193] = 194;

arr[194] = 195;

arr[195] = 196;

arr[196] = 197;

arr[197] = 198;

arr[198] = 199;

arr[199] = 200;

arr[200] = 201;

arr[201] = 202;

arr[202] = 203;

arr[203] = 204;

arr[204] = 205;

arr[205] = 206;

arr[206] = 207;

arr[207] = 208;

arr[208] = 209;

arr[209] = 210;

arr[210] = 211;

arr[211] = 212;

arr[212] = 213;

arr[213] = 214;

arr[214] = 215;

arr[215] = 216;

arr[216] = 217;

arr[217] = 218;

arr[218] = 219;

arr[219] = 220;

arr[220] = 221;

arr[221] = 222;

arr[222] = 223;

arr[223] = 224;

arr[224] = 225;

arr[225] = 226;

arr[226] = 227;

arr[227] = 228;

arr[228] = 229;

arr[229] = 230;

arr[230] = 231;

arr[231] = 232;

arr[232] = 233;

arr[233] = 234;

arr[234] = 235;

arr[235] = 236;

arr[236] = 237;

arr[237] = 238;

arr[238] = 239;

arr[239] = 240;

arr[240] = 241;

arr[241] = 242;</

each elementary size = 4 byte

classmate
Date _____
Page _____

⇒ 2D array =

↑
row
coloumn

datatype arrayname[x][y];

size → size of array.

eg → int arr[2][4];



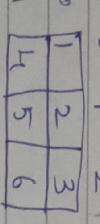
size of arr[2][4] = $2 \times 4 = 8$ elements.

Total size = ~~8+4~~ = $8+4 = 12$ bytes

* Initialization =

int arr[2][3] = {{1,2,3}, {4,5,6}}; 2x3 = 6 elements

int arr[2][3] = {{1,2,3}, {4,5,6}}, ① row, ② row,



* Size calculation =

can be calculated by multiplying the size of all the dimensions.

eg → size of arr[10][20] = $10 \times 20 = 200$ elements.

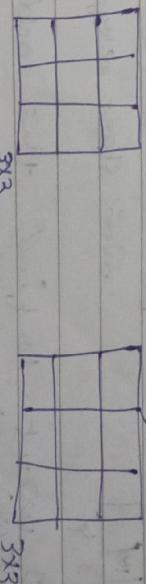
we can store 200 elements in this array.
Total size = $200 \times 4 = 800$ bytes

② size of arr[10][20] = $4 \times 10 \times 20 = 800$ bytes

= $800 \times 4 = 3200$ bytes

⇒ 3D array =

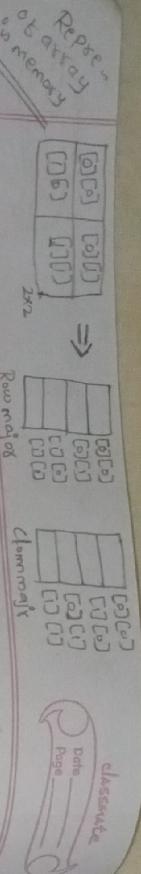
int arr[2][3][3],



* using program =
2D array elements can be printed using
2 nested for loop.

classmate
Date _____
Page _____

int arr[2][3] = {{1,2,3}, {4,5,6}},
for(i=0; i<2; i++) {
 for(j=0; j<3; j++) {
 printf("%d", arr[i][j]);
 }
}



* Initialization =

Put $a[2][2][3] = \{ \{ \{ 1, 2, 3 \}, \{ 4, 5, 6 \} \}, \{ 7, 8, 9 \}, \{ 10, 11, 12 \} \}$

(2×3)	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$
	2×3	2×3

* Using program =
3D array elements can be printed using

3 nested for loop.

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

 { { 7, 8, 9 }, { 10, 11, 12 } },

};

for (i=0; i<2; i++) {

 for (j=0; j<2; j++) {

 for (k=0; k<3; k++) {

 printf ("%d", a[i][j][k]);

 }

 }

}

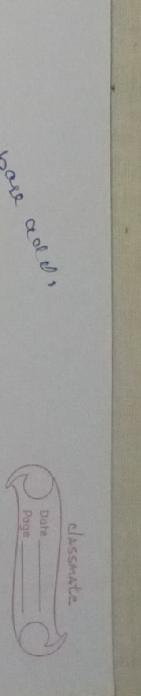
→ Representation of array in memory
Eq calculation =

eg → int a[5] $\begin{bmatrix} A & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \end{bmatrix}$
 $A[3] = 100$.

base address \rightarrow address - 2200
from start address addres.

Now I want to find the add (A[6])

$\text{Add}(A[6]) = \text{base add} + \text{index no} * \text{size of data type}$



$$\text{new } \rightarrow \text{want} \quad \text{Add}[i][j] = b + [i * n + j] * w. \quad \begin{array}{l} \text{Index form} \\ \text{[index form]} \end{array}$$

$$\begin{array}{l} \text{b} = \text{base add} \\ i = \text{row index} \\ j = \text{column index} \\ n = \text{columns} \\ w = \text{width} \end{array}$$

$$= 100 + [1 * 2 + 1] * 2 = 102$$

* Column major →

$$\text{Add}[i][j] = b + [j * m + i] * w \quad \begin{array}{l} \text{Index form} \\ \text{[index form]} \end{array}$$

$$\text{eg} \rightarrow \text{Add}[0][0] = 100 + [(0-1) * 2 + 0] * 2 = 100 + [1 * 2 + 0] * 2 = 102$$

$$\text{moreover } \quad \text{Add}[i][j] = b + [j * m + i] * w \quad \begin{array}{l} \text{Index form} \\ \text{[index form]} \end{array}$$

$$\text{eg} \rightarrow \text{Add}[0][0] =$$

$$100 + (0 * 2 + 0) * 2 =$$

$$100 + (0 * 2 + 1) * 2 =$$

$$100 + (0 * 2 + 2) * 2 =$$

$$100 + (0 * 2 + 3) * 2 =$$

$$100 + (0 * 2 + 4) * 2 =$$

$$100 + (0 * 2 + 5) * 2 =$$

$$100 + (0 * 2 + 6) * 2 =$$

$$100 + (0 * 2 + 7) * 2 =$$

$$100 + (0 * 2 + 8) * 2 =$$

$$100 + (0 * 2 + 9) * 2 =$$

$$100 + (0 * 2 + 10) * 2 =$$

$$100 + (0 * 2 + 11) * 2 =$$

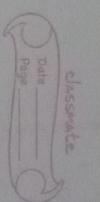
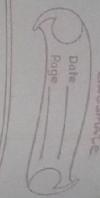
$$100 + (0 * 2 + 12) * 2 =$$

$$100 + (0 * 2 + 13) * 2 =$$

$$100 + (0 * 2 + 14) * 2 =$$

$$100 + (0 * 2 + 15) * 2 =$$

$$100 + (0 * 2 + 16) * 2 =$$



$$\begin{aligned} A[2][0] &= 10 + [j * m + l] * w \\ &= 100 + [1 * 3 + 2] * 2 \\ &= 100 + 10 = \underline{\underline{110}} \end{aligned}$$

\Rightarrow parallel arrays =

- * It is a "set" of an array which contains multiple arrays of the same size, in which i^{th} element of each array is related to each other.
- * All the elements in a parallel array represent a common entity.
- * E.g. Roll no -> 3rd element -> name. Marks -> 3rd element -> marks.
- * All the elements are related to each other.
- * Eg. → $\begin{bmatrix} 5 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}$ (A 2x3 matrix of 3 rows and 3 columns)

\Rightarrow sparse matrix =
A matrix is a 2D array made of non-zero elements in a column or row.

- * A matrix is → sparse matrix if most of its elements are 0.
- * A matrix is → sparse matrix if most elements are 0.
- * Eg. →

$$\begin{bmatrix} 5 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Row	0	1	2
Column	0	1	2
Value	5	3	2

Usage of a 2D array to represent a sparse matrix wastes a lot of memory because 0's in the matrix are useless in most cases.

- * A sparse matrix can be easily represented by a list of triplets of the form (Row, Column, Element).
- * We store only index of row, index of column & value of non-zero.

$$\begin{aligned} \text{eg. } &\rightarrow \begin{bmatrix} 5 & 3 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \Rightarrow \begin{array}{|c|c|c|c|c|} \hline \text{Row} & 0 & 0 & 1 & 2 \\ \hline \text{Column} & 0 & 1 & 3 & 2 \\ \hline \text{Value} & 5 & 3 & 1 & 2 \\ \hline \end{array} \end{aligned}$$

triple form, triplet form.

\rightarrow A[1] →

```

int marks[5] = {25, 20, 32, 30, 18};
for(i=0; i<n; i++) {
    if(marks[i] > max) {
        max = marks[i];
    }
}

```

```

int rollno[5] = {1, 2, 3, 4, 5};
int marks[5] = {25, 20, 32, 30, 18};

for(i=0; i<n; i++) {
    if(marks[i] > max) {
        max = marks[i];
    }
}

```

```

max = marks[i];
index = i;

```

{

printf("Rollno %d has highest marks",

rollno[index]);

? Action 0;

(3)