Arrays

The variable that we have used till now are capable of storing only one value at a time. Now consider a situation when we want to store and display the age of 100 employees. For that we have to do the following

1. Declare 100 different variables to store the age of Employee
2. Assign a value to each variable
3. Display the value of each variable

Although we can perform our task by the above three steps but just imagine how difficult it would be to handle so many variables in the program.

Therefore the concept of array was introduced.

An Array is a collection of similar type of data items and each data items is called an element of the array. The data type of the elements may be any valid data type like char, int or float. The elements of array share the same variable name but each element has a different index number known as subscripts.

Arrays can be simple dimensional or multidimensional. The Number of Subscripts determines the dimension of the array. A one dimension array has one ~~subsri~~ Subscript, two dimension has two or so many

The one dimension is known as vector & two dimension arrays are known as matrix.

One dimension Array

Like other variables, arrays should also be declared before they are used in the program.

Syntax

```
data-type  array_name [size];
```

→ Here array_name denotes the name of the array and it can be any valid C Identifier

→ Size of an array specifies the number of elements that can be stored in the array. It may be a positive integer constant or constant integer expression.

For Example

```
int age [100];
float sal [15];
char grade [20];
```

Here age is an integer type array, which can store 100 elements of type integer The array sal is a floating type array of size 15, can hold float values and the third one is a character type array of size 20, can hold characters.

as age[0], age[1], age[2], ------- age[99].
sal[0], sal[1], sal[2], ---- sal[14].
grade[0], grade[1], ------ grade[19].

when the array is declared, the compiler allocates spaces in memory sufficiently to hold all elements of the array, so the compiler should know the size of array at the compile time.

for Example

```
#include<stdio.h>
#define SIZE 10
int main()
{
  int size1 = 15;
  float sal[SIZE];   /* Valid */
  ____
  ____
}
```

Accessing 1-D Array elements

The element of an array can be accessed by specifying the array name followed by subscript in brackets.

NOTE:-

In C language array subscripts starts from 0. Hence if there is an array of size 5 then the valid subscripts will be from 0 to 4. Here last valid subscript (4) is also known as upper bound and the (0) is known as lower bound.

Let

int arr[5]; then.

arr[0], arr[1], arr[2], arr[3], arr[4]

Here 0 is lower bound.
4 is upper bound.

Processing 1-D Arrays.

for processing arrays we generally use a for loop variables is used at the place of subscripts. The initial value of loop variable is taken 0 since array subscripts starts from zero. The loop variable is increased by 1 each time.

Suppose arr[10] is an array of int type

(1) Reading values in arr[10]

```
for (i=0; i<10; i++)
    scanf ("%d", &arr[i]);
```

(2) Displaying the values of arr[10]

```
for (i = 0; i<10; i++)
    printf ("%d", arr[i]);
```

(3) Adding all the elements of arr[10]

```
sum = 0;
for (i=0; i<10; i++)
    sum = sum + arr[i];
```

Initilization of 1-D Array

Syntax

```
data_type array_name [size] = {value1, value2 ---
                               valueN};
```

⇒ data type of array.

⇒ array_name (name of array)

⇒ Size :- size of an array

⇒ value 1 --- ~~value~~ valueN are the constant value known as initializers, which are assigned to the array elements one after another.

for Example

1) int marks[5] = {50, 85, 70, 65, 95};

The values of the array after this initilization

marks[0] → 50
marks[1] → 85
marks[2] → 70
marks[3] → 65
marks[4] → 95

2) when initializing 1-D arrays, it is optional to specify the size of the array. if the size is omitted during initialization then compiler assumes the size of array equal to the number of initializers for Example.

int marks[] = {99, 78, 50};
float sal[] = {25.5, 38.5, 24.7};

Her the size of array marks and sal is 3.

3) If during initializing 1-D array number of initializers is less than the size of array then all the remaining elements of array are assigned value zero.

int marks[5]={99, 78};

Here size of array is 5 while there are only 2 initializers. Afterwords the elements of arrays are

marks[0] → 99

marks[1] → 78

marks[2] → 0

marks[3] → 0

marks[4] → 0

4) If we initialize an array like

```
int array[100] = {0};
```

Then all the elements will be initialized to zero.

*5) If the number of initilazed is more than given size given in brackets then compiler will shown an error

for Example.

```
int array[5] = {1,2,3,4,5,6,7,8};
```

Output

```
/* Error */
```

*** We cannot copy all the elements of an array to another array by simply assigning it to the other array.

for example

```
int a[5] = {1,2,3,4,5};
int b[5];
b=a;
```

→ Not valid.

for copy we have to use ~~ass~~ for loop.

```
for(i=0; i<5; i++)
    b[i] = a[i];
```

Two Dimensional Array

Syntax

```
data_type array_name [rowsize] [columnsize];
```

Here rowsize specify the numbers of rows and columnsize represents the number of columns in the array. The total number of elements in an arrays are rowsize * columnsize

for Example

```
int arr[2][2];
```

Here arr has 2 rows and 2 columns then total number of elements are

arr[2][2] ⇒ 2 × 2 ⇒ 4 Elements can be stored in an array

as.

Row \ col	0	1
0	arr[0][0]	arr[0][1]
1	arr[1][0]	arr[1][1]

Processing 2-D Arrays

for processing 2-D Arrays, we use nested for loops. The outer loop represents rows and inner for loop represents columns.

For Example

```
int arr[4][5];
```

(1) Reading values in arr

```
for(i=0; i<4; i++)
    { for(j=0; j<5; j++)
        { scanf("%d", &arr[i][j]);
        }
    }
```

(2) Displaying values of arr

```
for(i=0; i<4; i++)
    { for(j=0; j<5; j++)
        { printf("%d", arr[i][j]);
        }
    }
```

This will print all elements in the same line. If we want to print all the elements of different rows on different lines then we can write like this

```
for(i=0; i<4; i++)
    { for(j=0; j<5; j++)
        { printf("%d", arr[i][j]);
        }
      printf("\n");
    }
```

Initialization of 2-D Arrays

(1) 2-D Arrays can be initialized in a way similar to that of 1-D Arrays

for Example

```
int mat[4][3] = { 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22 };
```

These values are assigned to the elements row-wise, so the values of elements after this

Row →	Col 0	Col 1	Col 2
0	mat[0][0]	mat[0][1]	mat[0][2]
1	mat[1][0]	mat[1][1]	mat[1][2]
2	mat[2][0]	mat[2][1]	mat[2][2]
3	mat[3][0]	mat[3][1]	mat[3][2]

(2) while initializing we can group the elements row wise using inner braces. for example.

```
int mat[4][3] = { { 11, 12, 13 }, { 14, 15, 16 }, { 17, 18, 19 },
                  { 20, 21, 22 } };
```

(Row 0) (Row 1) (Row 2) (Row 3)

OR

```
int mat[4][3] = {
                  { 11, 12, 13 };   → Row 0
                  { 14, 15, 16 },   → Row 1
                  { 17, 18, 19 },   → Row 2
                  { 20, 21, 22 }    → Row 3
                };
```

(3) Now consider this array.

```
int mat[4][3] =
                {
                  {11},          —— Row 0
                  {12,13},       —— Row 1
                  {14,15,16},    —— Row 2
                  {17}           —— Row 3
                };               —— End.
```

The remaining elements in each row will be assigned as 0 value. as.

mat[0][0] : 11	mat[0][1] : 0	mat[0][2] : 0
mat[1][0] : 12	mat[1][1] : 13	mat[1][2] : 0
mat[2][0] : 14	mat[2][1] : 15	mat[2][2] : 16
mat[3][0] : 17	mat[3][1] : 0	mat[3][2] : 0

(4) In 2D Arrays it is optional to specify the first dimension but the second dimension should be present.

for Example

```
int mat[][3] = {
                 {1,10},
                 {2,20,200},
                 {3},
                 {4,40,600}
               };
```

Here first dim is taken as 4 since there are 4 rows.

Address Calculation in Single dimension Array :-

Actual Address of the 1st Element of the Array known as Base Address (B)

(Here B is 1100)

*Memory space width (W) (4 bytes)

Actual Address in the Memory	1100	1104	1108	1112	1116
Elements	15	7	11	44	93
Address with respect to the Array subscript	0	1	2	3	4

↑ Lower limit/ Bound of subscript (LB)

* Memory space acquired by every element in the Array is called width (W)
  Here it is of 4 bytes

formula

Address of A[I] = B + W * (I - LB)

B = Base Address
W = Storage size of one element stored in the array (in bytes)
I = Subscript of a element whose address

is to be found.

LB = Lower limit / Lower bound of subscript if not specified assumed to zero 0.

Example

Given the BASE Address of an array B[1300....1900] as 1020 and size of each element is 2 bytes in the memory then find the address of B[1700].

Solution :-

The given values are

$B = 1020$, $LB = 1300$, $W = 2$, $I = 1700$

Then Address of B[1700] is

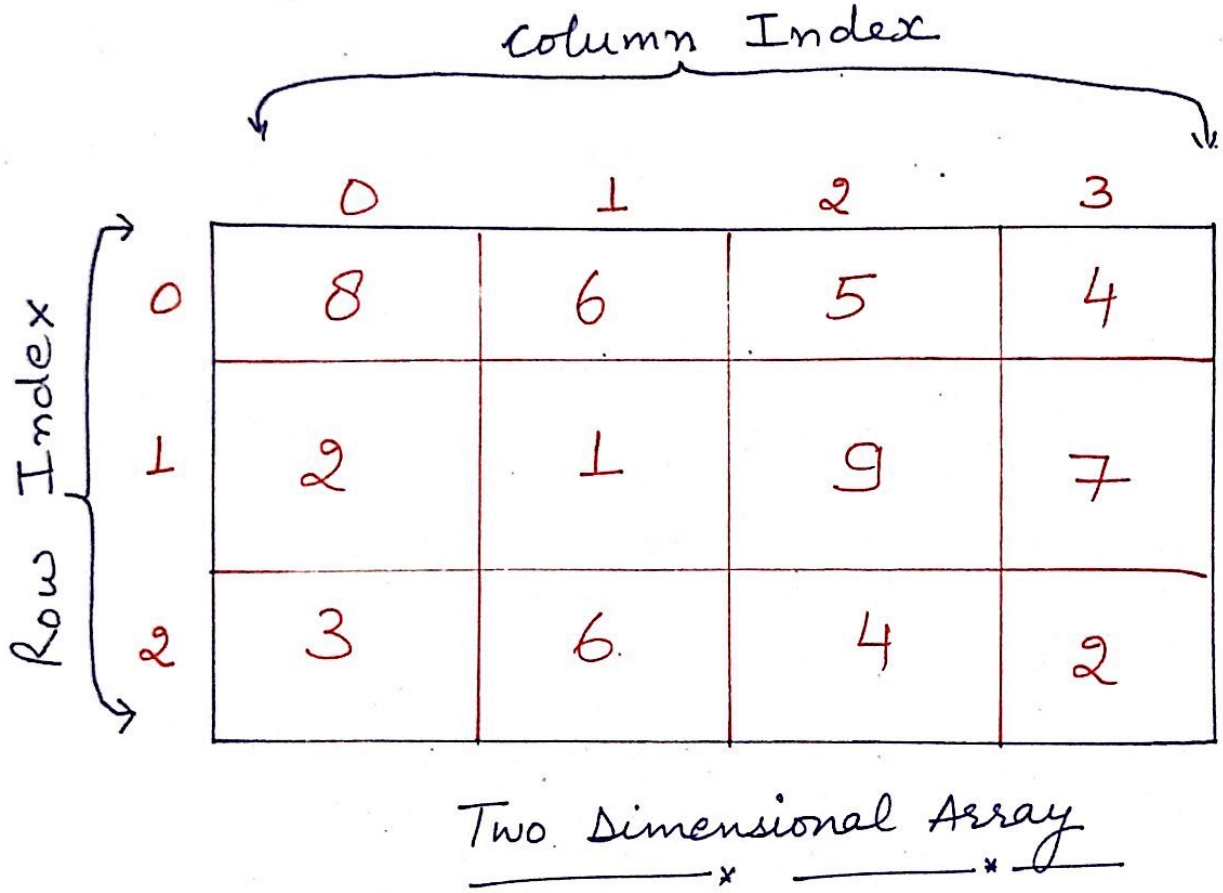
$$B[1700] = B + W * (I - LB)$$
$$= 1020 + 2 * (1700 - 1300)$$
$$= 1020 + 2 * 400$$
$$= 1020 + 800$$
$$= 1820 \quad \text{Ans}$$

Address calculation in Double (Two) dimensional Array :-

while sorting the elements of a 2-D Array in memory, these are allocated contiguous memory locations. Therefore a 2-D Array must be linearized so as to enable their storage.

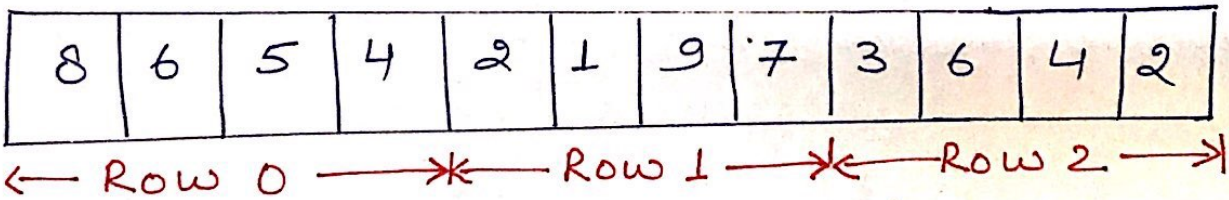
There are two alternative solution for that.

1. Row-Major
2. Column Major

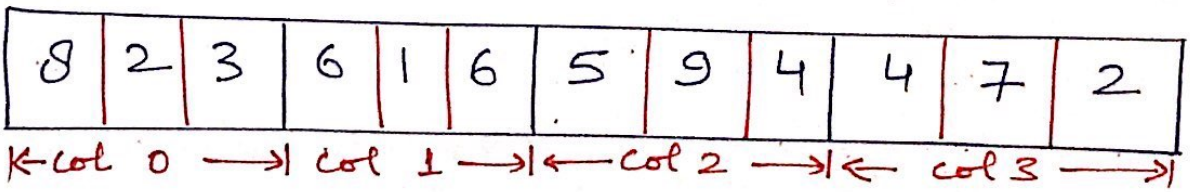


Two Dimensional Array

① Row-Major (Row wise Arrangement)



② Column-Major (Column wise Arrangement)



① Row Major

The Address of a location in Row major system is calculated using the following formula. Let we have to calculate the Address of A[i][j] array. then.

$$\text{Address } A[i][j] = B + W * [N * (i - Lr) + (j - Lc)]$$

B = Base Address
W = storage size of one element stored in the array (in byte)
N = No of columns of the given matrix.
i = Row subscripts of element whose address is to be found.
Lr = Lower limit of row / start row index of matrix, if not given assume 0.
j = column subscript of element whose address is to be found.
Lc = Lower limit of column / start column index of matrix. if not given assume zero(0).

② Colum Major

Let we have to calculate the Address of A[i][j] array then.

$$\text{Address of } A[i][j] = B + W * [(i - Lr) + M * (j - Lc)]$$

B = Base Address
W = storage size of one element stored in the array (in bytes)

i = Row subscripts of Element whose address is to be found

L_r = Lower limit of row / start row index of matrix. If not given Assume zero.

M = No of rows of the given matrix.

j = column subscripts of Element whose address is to be found.

L_c = Lower limit of column / start column index of matrix. If not given assume zero.

IMP:-

Usually number of rows and columns of a matrix are given (like A[20][30]). But if it is given as $A[L_r \text{-----} U_r, L_c \text{-----} U_c]$. In this case number of rows and columns are calculated using the following methods

⇒ Number of rows (M) will be calculated as =

$$(U_r - L_r) + 1$$

⇒ Number of columns (N) will be calculated as =

$$(U_c - L_c) + 1$$

And rest process will remain same as per requirement (Row Major or Column Major)

For Example

Ques 1

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

Solution

No of rows say $M = (U_r - L_r) + 1$

$= [10 - (-15)] + 1$

$= (10 + 15) + 1$

$= 25 + 1$

$M = 26$

No of column say $N = (U_c - L_c) + 1$

$= (40 - 15) + 1$

$= 25 + 1$

$N = 26$

(1) Column Major

The given values are $B = 1500$, $W = 1$ bytes, $I = 15$, $L_c = 15$, $M = 26$, $J = 20$

Address of X[15][20] $= B + W * [(I - L_r) + M * (j - L_c)]$

$= 1500 + 1 * ([15 - (-15) + 26 * (20 - 15)]$

$= 1500 + 1 * (30 + (26 * 5))$

$X[15][20] = 1660$ Ans.

(2) Row Major

The given values are $B = 1500$, $W = 1$ bytes, $I = 15$, $J = 20$, $L_r = 15$, $L_c = 15$, $N = 26$

Address of X[15][20]

$= B + W * [N * (i - Lr) + (j - Lc)]$

$= 1500 + 1 * [26 * (15 - (-15)) + (20 - 15)]$

$= 1500 + 1 * [(26 * 30) + 5$

$= 1500 + 1 * [780 + 5]$

$= 1500 + 785$

X[15][20] = 2285 Ans

More Examples

① Concider 30X4 2D Array and base address is 200 and 1 word per memory locations find out the address of A[15][3].

Solution

BA = 200, M = 30, N = 4, i = 30, j = 4

W = 1

① Row-wise :—

Location of A[15][3] = $B + W * [N * (i - Lr) + (j - Lc)]$

$= 200 + 1 * [4 * (15 - 1) + (3 - 1)]$

$= 200 + 1 * [(4 * 14) + 2]$

$= 200 + 1 * (56 + 2)$

$= 200 + 58.$

$= 258$ Ans.

(2) Column wise :-

Location of $A[15][3] = B * W * [(i - L_r) + M * (j - L_c)]$

$= 200 + 1 * [(15 - 1) + 30 * (3 - 1)]$

$= 200 + 1 * [14 + 30 * 2]$

$= 200 + 1 * [14 + 60]$

$= 200 + 1 * 74$

$= 274$ Ans