Array

Algorithm to search a particular element from a list

```
1. Input: A set of data in array a, and variable x. i.e., the target element
a[1….n], x;
2. found=0
3. for (i=1; i<=n; i++)
{
        if (a[i]==x)
        location=i;
        found= 1;
        break;
}
4. Output: if (found==1)
                print”FOUND” message and location.
                else print ”NOT FOUND” message
```

Algorithm to find out the summations of even and odd numbers

```
1. Input: An array and variable (to store the results of summations)
        A[1….n], sum_odd=0, sum_even=0;
2. for (i=1; i<=n; i++)
{
        if(A[i]%2==0), sum_even=sum_even+A[i];
        else sum_odd=sum_odd+A[i];
}
3. Output: Summationof odd numbers (print sum_odd) and
        Summation of even numbers (print sum_even)
```

Algorithm to find out the summations of even and odd indexed numbers

```
1. Input: An array and variable (to store the results of summations)
        A[1….n], sum_odd=0, sum_even=0;
2. for (i=1; i<=n; i++)
{
        if(i%2==0), sum_even=sum_even+A[i];
        else sum_odd=sum_odd+A[i];
}
3. Output: Summation of numbers in odd indices (print sum_odd) and
        Summation of numbers in even indices (print sum_even)
```

Algorithm to insert an element into an array

```
1. Input: An array A[1….n], the position of insertion m and the data x.
2. Increase the size of the array, A[1….n+1]
3. for (i=n; i>=m; i--)
        A[i+1]=A[i];
4. A[m]=x;
5. Output: The array, A with size n+1
```

Algorithm to delete an element from an array

```
1. Input: An array A[1….n], the position of deletion m.
2. for (i=m; i<n; i++)
        A[i]=A[i+1];
3. Output: The updated array, A
```

Two Dimensional Array

Definition

Two dimensional array is an array that has two dimensions, such as row and column. Total number of elements in a two dimensional array can be calculated by multiplication of the numbers of rows and the number of columns. If there are m rows and n columns then the total number of elements is mxn, and mxn is called the size of the array. Of course, the data elements of the array will be same type.

In mathematics, the two dimensional array can be expressed as follows:

A_{ij} or A[I,j] for 1<=i<=m and 1<=j<=n (where m and n are the number of rows and columns respectively)

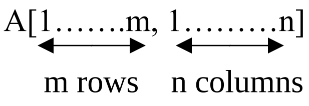
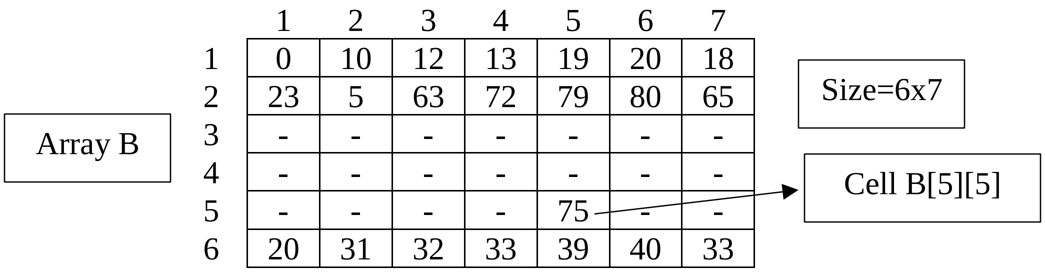


Fig 1: Symbolic representation of two dimensional array



	1	2	3	4	5	6	7
1	0	10	12	13	19	20	18
2	23	5	63	72	79	80	65
3	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-
5	-	-	-	-	75	-	-
6	20	31	32	33	39	40	33

Fig 2: Graphical representation of two dimensional array

Store and retrieve values in and from array

```
for(i=0; i<2; i++) {
    for(j=0;j<3;j++)
      scanf("%d", &B[i][j]);
  }
  for(i=0; i<2; i++) {
    for(j=0;j<3;j++)
      printf("%d ", B[i][j]);
    }
```

Two dimensional array representation in memory

The elements of a two dimensional array are stored in computers memory row by row or column by column. If the array is stored as row by row, it is called row-major order and if the array is stored as column by column, it is called column-major order.

In row-major order, elements of a two dimensional array are stored as-

$[A_{11}, A_{12}, A_{13}, A_{14}, A_{15}, A_{16}]$, $[A_{21}, A_{22}, A_{23}, A_{24}, A_{25}, A_{26}]$,……$[A_{51}, A_{52},….A_{56}]$

row 1 row 2 row 5

And in column-major order, elements are ordered as-

$[A_{11}, A_{21}, A_{31}, A_{41}, A_{51}, A_{61}]$, $[A_{12}, A_{22}, A_{32}, A_{42}, A_{52}, A_{62}]$,……$[A_{16}, A_{26},….A_{56}]$

column 1 column 2 column 6

	1st	2nd	3rd	4th	5th	6th	
	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	
7th	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	A_{26}	
13th							
						A_{46}	24th
25th	A_{51}	A_{52}	A_{53}	A_{54}	A_{55}	A_{56}	
		26th	27th	28th	29th	30th	

(a) Row-major order

	1st	6th	11th				
	A_{11}	A_{12}	A_{13}	-	-	A_{16}	26th
2nd	A_{21}	A_{22}	-	-	-	A_{26}	27th
3rd	A_{31}	A_{32}	-	-	-	A_{36}	28th
4th	A_{41}	A_{42}	-	-		A_{46}	29th
5th	A_{51}	A_{52}	-	-	A_{55}	A_{56}	30th
					25th		

(b) Column-major order

Fig. 3: Pictorial view of two dimensional array representation in memory.

Location of an element of a two-dimensional array

Row-major Order

If LOC(A[i,j]) denotes the location in the memory of the element A[i,j] or A_{ij}, the in row-major order-

LOC(A[i,j])=Base(A)+(n(i-1)+(j-1)))*w;

Here Base(A) is the starting or base address of the array A, n is the number of columns and w is the width of the cell, i.e., number bytes per cell.

Column-major Order

In column-major order,

LOC(A[i,j])=Base(A)+(m(j-1)+(i-1)))*w;

Here Base(A) is the starting or base address of the array A, m is the number of rows and w is the cell width.

Example

Base address, Base(A)=100, Size of the array =5x6. If the type of array is integer then find out LOC(A[4,3]).

Solution:

If the array is stored in row-major order:

LOC(A[4,3])=Base(A)+(n(i-1)+(j-1)))*2 (2 bytes for each integer cell in C/C++)

=100+(6(4-1)+(3-1))*2

=100+(6x3+2)*2

=100+(18+2)*2

=100+(20)*2

=100+40

=140

If the array is stored in column-major order:

LOC(A[4,3])=Base(A)+(m(j-1)+(i-1)))*2

=100+(5(3-1)+(4-1))*2

=100+(5x2+3)*2

=100+(10+3)*2

=100+(13)*2

=100+26

=126

Algorithm to find out the summation of boundary elements

```
1. Input: a two-dimensional array
        A[1….m,1….n]
        Sum=0;
2. Find each boundary element
        for(i = 1; i <= m; i++)
                for(j = 1; j <= n; j++)
                        if(i = 1 || j = 1 || i = m || j = n), sum=sum+A[i.j],
3. Output: Print sum as the result of summation of boundary elements
```

Algorithm to find out the summation of diagonal elements

```
1. Input: a two-dimensional array
        B[1….n,1….n]
        Sum=0;
2. Find each diagonal element and add them with sum
        for(i = 1; i <= n; i++)
                for(j = 1; j <= n; j++)
                        if(i = j || i+j = n+1), sum=sum+B[i.j],
3. Output: Print sum as the result of summation of diagonal elements
```