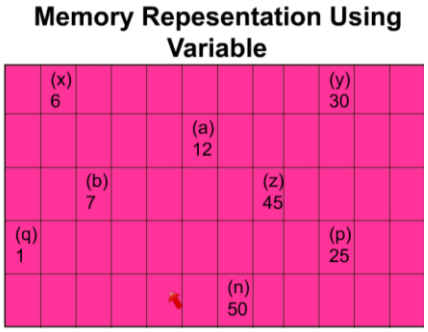
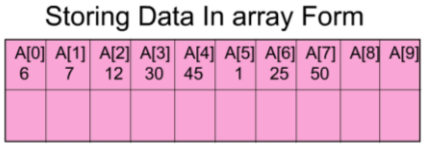
Array Notes (Class XI)

Array - Array is a finite number of data elements of the same type. Array elements are stored in consecutive memory cells under a single variable name in a single row or in single column or in rows and columns both.

Advantage - We all know that computer memory has unique address for each of its memory cells and these memory locations are accessed at random by using these variables. Thus in order to store a large number (100,1000 or more) of data we require large number of variables. Using array these can be stored in consecutive memory location and manipulation or accessibility will be easier. Example : Arrays can be used to store details of the employees, cities etc.





Types of Array

Arrays are classified according to the manner in which data are stored. When data are stored in consecutive memory in a single row or column is called **Single Dimensional Array**. When data is stored in rows and columns i.e. in matrix form then it is called **double dimensional array**. Multi Dimensional array has more than two dimensions.

Accessing Data from Single Dimension Array -Once data are stored in consecutive memory locations into a single dimensional array , they can be accessed by referring to their respective position. The position of the first element is considered as position 0 and the position of last element will be n-1 position.

Defining An Array

Syntax: Data type Array Name [index]

int num[50]; // can store maximum 50 integer type data

char alpha[26]; // can store maximum 26 characters

Note : size of array should be a finite positive integer.

Array Operations

1. Accept 2. Display 3. Traversal 4. Processing /replacing array elements 5. Swapping elements

Character Array Operations

1D Character Array – To store name, city name etc.

Defining – char name[80];

Accept – cin>>name ; or gets(name);

Display – cout<<name;

Initialization – char name[]=”Computer Science”; //entire string

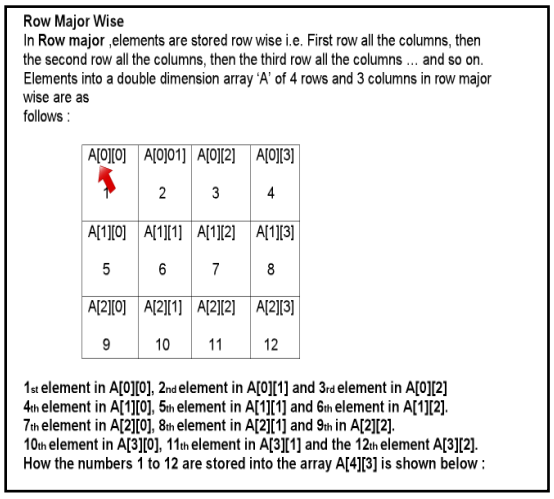
char pass[5]={‘p’,’a’,’s’,’s’,’\0’}; // character by character

Accessing element by element – String ends with ‘\0’ .

for(int i=0;name[i]!=’\0’;i++) cout<<name[i]<<”\n”; will display each character in individual lines.

Functions related to string array

Double Dimension Array - In a double dimension array elements are arranged in rows and columns and we need two subscripts , one each for row and column. Elements can be stored into a double dimension array in two ways: 1)Row Major wise and 2) column major wise

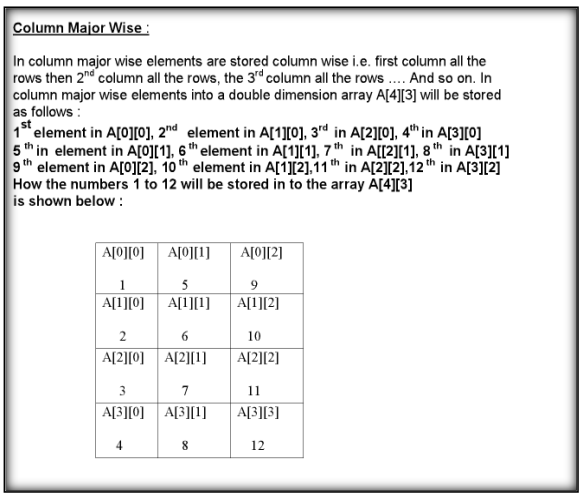


Row Major Wise

In **Row major** ,elements are stored row wise i.e. First row all the columns, then the second row all the columns, then the third row all the columns ... and so on. Elements into a double dimension array ‘A’ of 4 rows and 3 columns in row major wise are as follows :

A[0][0] 1	A[0]01] 2	A[0][2] 3	A[0][3] 4
A[1][0] 5	A[1][1] 6	A[1][2] 7	A[1][3] 8
A[2][0] 9	A[2][1] 10	A[2][2] 11	A[2][3] 12

1st element in A[0][0], 2nd element in A[0][1] and 3rd element in A[0][2]
4th element in A[1][0], 5th element in A[1][1] and 6th element in A[1][2].
7th element in A[2][0], 8th element in A[2][1] and 9th in A[2][2].
10th element in A[3][0], 11th element in A[3][1] and the 12th element A[3][2].
How the numbers 1 to 12 are stored into the array A[4][3] is shown below :



Column Major Wise :

In column major wise elements are stored column wise i.e. first column all the rows then 2nd column all the rows, the 3rd column all the rows …. And so on. In column major wise elements into a double dimension array A[4][3] will be stored as follows :

1st element in A[0][0], 2nd element in A[1][0], 3rd in A[2][0], 4th in A[3][0]
5th in element in A[0][1], 6th element in A[1][1], 7th in A[[2][1], 8th in A[3][1]
9th element in A[0][2], 10th element in A[1][2],11th in A[2][2],12th in A[3][2]
How the numbers 1 to 12 will be stored in to the array A[4][3]
is shown below :

A[0][0] 1	A[0][1] 5	A[0][2] 9
A[1][0] 2	A[1][1] 6	A[1][2] 10
A[2][0] 3	A[2][1] 7	A[2][2] 11
A[3][0] 4	A[3][1] 8	A[3][3] 12

Example Matrix is of 3 x 3 and contains following values: 1 2 3 4 5 6 7 8 9	In the form of index (i=row , j= column) i j 0 1 2 0 00 01 02 1 10 11 12 2 20 21 22
Accept Array	Display all elements in Array

```
void accept( int x[3][3])
{
 for (int i=0;i<3;i++)
{ cout<<"Enter elements in row "<<i+1;
for(int j=0;j<3;j++)
{ cin>>x[i][j];
}
}
}
```

```
void display( int x[3][3])
{
 for (int i=0;i<3;i++)
{ cout<<"elements in row "<<i+1<<"\n";
for(int j=0;j<3;j++)
{ cout<<x[i][j]<<"  ";
}
}
}
```

3.Display the elements in pattern

```
1
4 5
7 8 9
void display_pat1( int x[3][3])
{
 for (int i=0;i<3;i++)
{ cout<<"elements in row "<<i+1<<"\n";
for(int j=0;j<=i;j++)
{ cout<<x[i][j]<<"  ";
}
}
}
```

4.Display the elements in pattern

```
1 2 3
4 5
7
void display_pat2( int x[3][3])
{
 for (int i=0;i<3;i++)
{ cout<<"elements in row "<<i+1<<"\n";
for(int j=0;j<3-i;j++)
{ cout<<x[i][j]<<"  ";
}
}
}
```

5. Display the elements in pattern(ignoring space)

```
        3
    5   6
 7  8   9
void display_pat3( int x[3][3])
{
 for (int i=0;i<3;i++)
{ cout<<"elements in row "<<i+1<<"\n";
for(int j=3-1-i;j<3;j++)
{ cout<<x[i][j]<<"  ";
}
}
}
```

6..Display the elements in pattern(ignoring space)

```
 1  2  3
    5  6
       9
void display_pat4( int x[3][3])
{
 for (int i=0;i<3;i++)
{ cout<<"elements in row "<<i+1<<"\n";
for(int j=i;j<3;j++)
{ cout<<x[i][j]<<"  ";
}
}
}
```

7. Display the diagonal element

```
 1
    5
       9
void display_diag1( int x[3][3])
{
 for (int i=0;i<3;i++)
{ cout<<"elements in row "<<i+1<<"\n";
for(int j=0;j<3;j++)
```

8. Display the diagonal elements

```
        3
    5
 7
void display_diag1( int x[3][3])
{
 for (int i=0;i<3;i++)
{ cout<<"elements in row "<<i+1<<"\n";
for(int j=0;j<3;j++)
```

{if(i==j) cout<<x[i][j]<<" "; } } }	{if(i+j==3-1) cout<<x[i][j]<<" "; } } }
9. Sum of elements as per any of the problems given below , replace cout statement with s=s+x[i][j];	10. Processing array elements like skip elements , or conversion of 1D to 2D / 2D to 1D
11. Array Initialisation int x[3][3]= {{1,2,3},{4,5,6},{7,8,9}};	