Multi-Dimensional Array

A multi-dimensional array is an array with more than one level or dimension. It might be 2-Dimensional and 3-Dimensional and so on.

Consider the example of storing marks of 5 students in some particular subject A. We can define,

int A_marks[ ] = {90,95,99,100,89};

Consider the example of storing marks of 5 students in 3 different subjects A, B and C. We can say,

Version-1:

int A_marks[ ] = {90,95,99,100,89};

int B_marks[ ] = {95,90, 91,99, 94};

int C_marks[ ] = {91,92,93,95,100};

Version-2:

int student_1_marks[ ] = {90, 95, 91};

int student_2_marks[ ] = {95, 90, 92};

int student_3_marks[ ] = {99, 91, 93};

int student_4_marks[ ] = {100, 99, 95};

int student_5_marks[ ] = {89, 94, 100};

Now, rather than storing marks this way in One - Dimensional Array, we can use Two - Dimensional array as we have two dimensions involved in this particular example: student and subject marks

Two – Dimensional Array

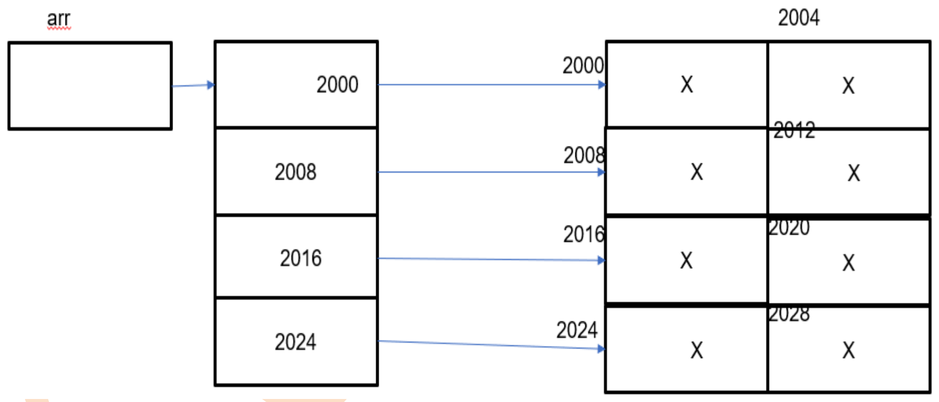
It is treated as an array of arrays. Every element of the array must be of same type as arrays are homogeneous. Hence, every element of the array must be an array itself in 2D array.

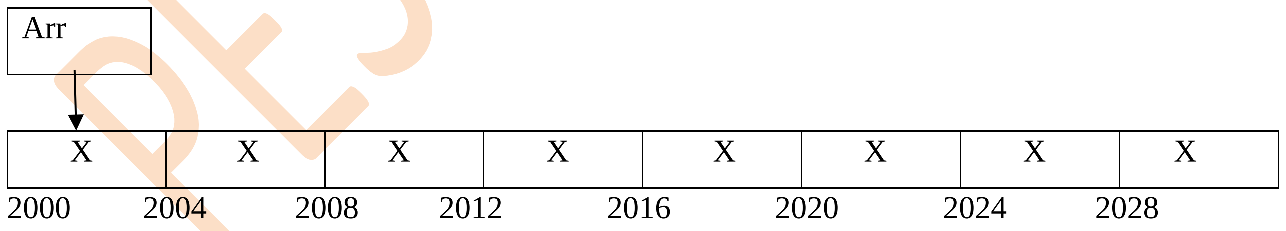
Let us consider the above example of storing 5 students marks in three different subjects. int marks[5][3] = {{90, 95, 91}, {95, 90, 92}, {99, 91, 93}, {100, 99, 95}, {89, 94, 100} };

Declaration of a 2D Array

data_type array_name[size_1][size_2];

int arr[4][2]; // Allocate 8 contiguous memory locations





If the size of the integer is 4 bytes ,8 contigeous memory allocated

Initialization of a 2D Array: Compiler knows the array size based the array elements and allocates memory

data_type array_name[size_1][size_2] = {elements separated by comma};

int arr[][] = {11,22,33,44,55,66}; // Error. Column size is compulsory

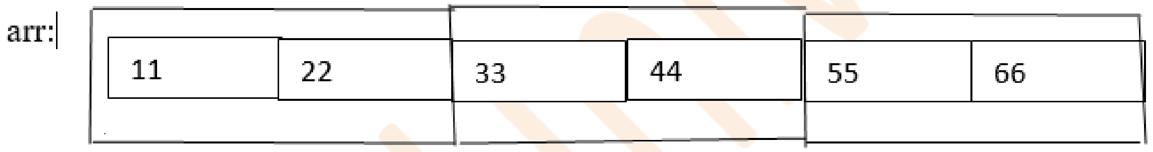
Without knowing the number of columns in each row – it is impossible to locate an element in 2D array.

int arr[3][2] = {{11,22},{33,44},{55,66}};

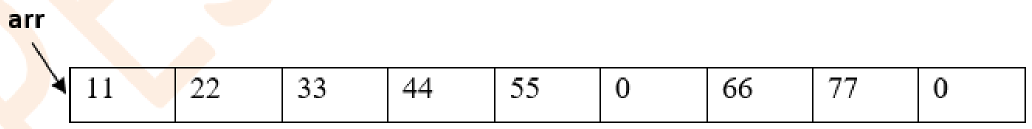
int arr[3][2] = {11,22,33,44,55,66};

// Allocate 6 contiguous memory locations and assign the values

Int arr[][2] = {11,22,33,44,55,66};



int arr[][3] = {{11,22,33},{44,55},{66,77}};   **//Partial initialization**

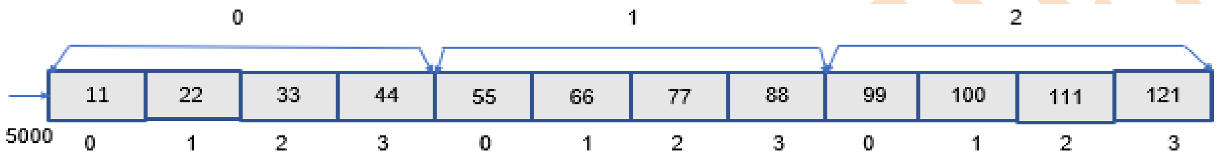


Internal Representation of a 2D Array

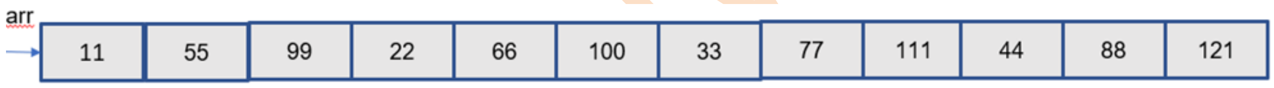
As two - dimensional array itself is an array, elements are stored in contiguous memory locations. And since elements are stored in contiguous memory locations, there are two possibilities.

int arr[3][4] = {{11, 22, 33, 44}, {55, 66, 77, 88}, {99, 100, 111, 121}};

Row major Ordering: All elements of one row are stored followed by all elements of the next row and so on.



Column Major Ordering: All elements of one column are stored followed by all elements of the next column and so on.



Generally, systems support Row Major Ordering.

<u>**Address of an element in a 2D Array**</u>

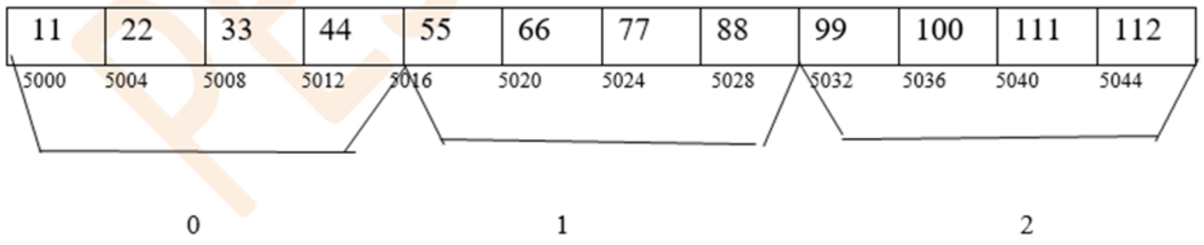
Address of A[i][j] = Base_Address +((i * No. of columns in every row) + j)*size of every element;

Consider, int arr[3][4] = {{11, 22, 33, 44}, {55, 66, 77, 88}, {99, 100, 111, 121}};

If the size of integer is 4 bytes in the system and the base address is 5000, then address of arr[1][2] can be found by 5000+((1*4)+2)*4 = 5000+6*4 = 5000+24 = 5024.

Consider, int arr[3][4] = {11, 22, 33, 44, 55, 66, 77, 88, 99, 100, 111, 121};

If the size of integer is 4 bytes in the system and the base address is 5000, then address of arr[1][5] can be found by 5000+((1*4)+5)*4 = 5000+9*4 = 5000+36 = 5036.



Code to Read and display 2D Array

```
int a[100][100];

int n; int m;

printf("enter row num and column numbers\n");

 scanf("%d %d",&n,&m);

printf("Enter %d elements",n*m);

for(int i = 0;i<n;i++)   // n – row index          // m – column index

{

        for(int j= 0;j<m;j++)

        {

                scanf("%d",&a[i][j]);

        }

}
```

```
printf("entered elements are\n");

for(int i = 0;i<n;i++)

{

        for(int j= 0;j<m;j++)

        {

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

        }

        printf("\n");

}
```

Two-Dimensional Array and Pointers

Consider int a[5][2] = {{11,22}, {33,44}, {55,66}, {77,88}, {99,100}};

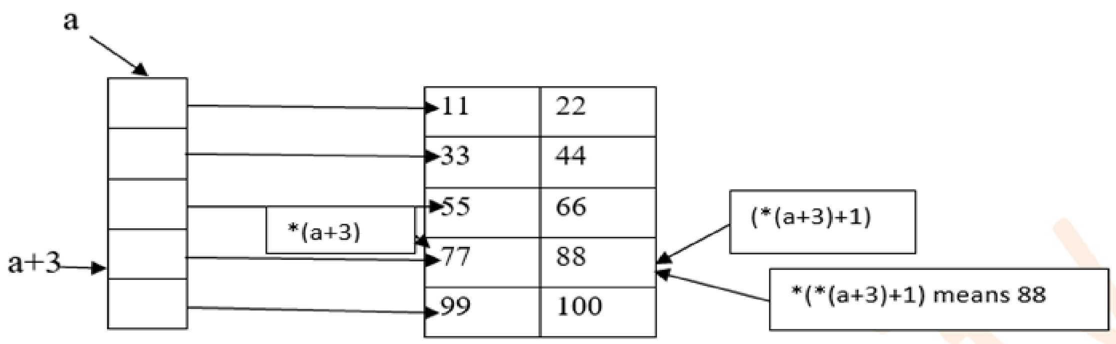
```
        printf("Using array, accessing the elements of the array");

        printf("%d\n",a[3][2]);         // 99   array notation

        printf("%d\n",a[3][1]);         // 88    array notation

        printf("%d\n",*(*(a+3)+1)); // 88      // pointer notation
```

Array name is a pointer to a row. The expression a + 3 is a pointer to the third row. *(a + 3) becomes a pointer to the zeroth element of that row. Then +1 becomes a pointer to the nextelement in that row. To get the element in that location, dereference the pointer.



In General, (a + i) points to ith 1-D array

//int *p1 = a;// incompatible type. Warning during compilation

// If above statement is fine, what happens when p1[7] is accessed and p1[2][1] is accessed

int (*p)[2] = a;// **p is a pointer to an array of 2 integers**

Since subscript([ ]) have higher precedence than indirection(*), it is necessary to have parentheses for indirection operator and pointer name. Here the type of p is 'pointer to an array of 2 integers'.

printf("using pointer accessing the elements of the array")

printf("%d\n",p[3][1]);   // 88   array notation

printf("%d\n",*(*(p+3)+1)); // 88   pointer notation

Note : The pointer that points to the 0th element of array and the pointer that points to the whole array are totally different.

Consider the program to illustrate the size of pointer to an array of items.

```
int arr[4][3]={33,44,11,55,88,22,33,66,99,11,80,9};

//assigning array to a pointer

int *p=arr;                    //gives warning

//printf("%d",p[7]);  //looks fine,not a good idea.

//printf("\n%d\n",p[2][1]);          //error..Column size not available to p.

//create a pointer to an array of integers

int (*p)[3]=arr;

printf("%d\n",p[2][1]);

printf("%d\n",*(*(p+2)+1));

printf("%d\n",sizeof(p));      //size of pointer

printf("%d %d\n",sizeof(*p),sizeof(arr));       //size of the array pointing to 0th row and size of the entire 2D array

}
```

Passing 2D Array to a function

Let us try to read and display 2D array using functions. Client code is as below.

```
int main()

{

int a[100][100];

int m1,n1;

printf("enter the order of a\n");

scanf("%d%d",&m1,&n1); // user entered 3 5

printf("enter the elements of a\n");

read(a,m1,n1);

printf("elements of A are\n");

display(a,m1,n1);

}
```

Consider the following cases while passing 2D array to functions

Case1: Declaration and definition of the function with no column size

```
        void read(int[][],int m,int n);      //Compiletime error

        void display(int[][],int m,int n);
```

Case 2: Declaration and definition of the function with column size not same as what is given in the declaration of the array in the client code

```
void read(int[][3],int m,int n);        // warning but code works

void display(int[][3],int m,int n);
```

Case 3: Declaration and definition of the function with column size same as what is given in the declaration of the array in the client code

```
void read(int[][100],int m,int n);        // fine.

void display(int[][100],int m,int n);
```

// But can we say void read(int[][n],int m,int n) ? Think !!!!

Case 4: If the order of parameters is changed as below in declaration and definition of the function, we need to change the client code. Changing the client code is not a good programmers habit. Because interface cannot be changed. Implementation can change. So refer to case 5.

```
void read(int m,int n,int a[][n]);

void display(int m,int n,int a[][n]);
```

Case 5: Forward declaration

```
void read(int n;int[][n],int m,int n);

void display(int n; int[][n],int m,int n);
```

```
void read(int n;int a[][n],int m,int n)

{

int i,j;
```

```
          for(i = 0;i<m;i++)

                    for(j = 0;j < n;j++)

                    {

                              printf("enter the element");

                              scanf("%d",&a[i][j]);

                    }

}

void display(int n; int a[][n],int m,int n)

{

          int i,j;

          for(i = 0;i<m;i++)

          {

                    for(j = 0;j < n;j++)

                    {

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

                    }

                    printf("\n");

          }

}
```

Case 6: Array degenerates to a pointer at run time. So how about using pointer to an array as a parameter.

```
                    void read(int n; int(*)[n],int m,int n);

                    void display(int n; int(*)[n],int m,int n);

void read(int n; int (*a)[n],int m,int n)
{
          int i,j;
          for(i = 0;i<m;i++)
                    for(j = 0;j < n;j++)
                    {
                              printf("enter the element");
                              scanf("%d",&a[i][j]);
                    }
}
void display(int n; int (*a)[n],int m,int n)
{
          int i,j;
          for(i = 0;i<m;i++)
          {
                    for(j = 0;j < n;j++)
```

```
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
}
```

Let us write a function to add, subtract and multiply two matrices. Display appropriate message when these two matrices are not compatible for these operations.

```
void read(int (*a1)[],int,int); void display( int (*a1)[],int,int);

void multiply(int n1; int n2; int (*a)[n1],int (*b)[n2],int m1,int n1,int n2,int (*c)[n2]);

 void subtract(int n; int (*a)[n],int (*b)[n],int m, int n, int (*c)[n]);

void add(int n; int (*a)[n],int (*b)[n],int m, int n, int (*c)[n]);

 int main()

{

        int a[100][100];

        int b[100][100];

        int c[100][100];

        int m1,n1;
 int m2,n2;

printf("enter the order of a\n");

scanf("%d%d",&m1,&n1);

 printf("enter the elements of a\n");

read(a,m1,n1);

printf("enter the order of b\n");

scanf("%d%d",&m2,&n2);

printf("enter the elements of b\n");

 read(b,m2,n2);

if (m1 == m2 && n1 == n2)

{

            printf("elements of A are\n");

            display(a,m1,n1);

            printf("elements of B are\n");

            display(b,m2,n2);

            printf("Addition of two matrices\n");

            add(a,b,m1,n1,c);

            display(c,m2,n2);

            printf("Subtraction of two matrices\n");

            subtract(a,b,m1,n1,c);
```

```
        display(c,m2,n2);
    }
    else
    {
        printf("Two matrices are not compatible for additiona nd subtraction\n");
    }

    if (n1==m2)
    {
        printf("Multiplication of two matrices\n");
        multiply(a,b,m1,n1,n2,c);
        display(c,m1,n2);
    }
    else
    {
        printf("Two matrices are not compatible for multiplication too\n");
    }
    return 0;
}
```

```
 void add(int n; int (*a)[n],int (*b)[n],int m, int n, int (*c)[n])
{
        for(int i = 0;i<m;i++)
        {
                for(int j= 0;j<n;j++)
                {
                        c[i][j] = a[i][j]+b[i][j];
                }
        }
}
void subtract(int n; int (*a)[n],int (*b)[n],int m, int n, int (*c)[n])
{
        for(int i = 0;i<m;i++)
        {
                for(int j= 0;j<n;j++)
                        {
                                c[i][j] = a[i][j] - b[i][j];
                        }
        }
```

```
}
void multiply(int n1; int n2; int (*a)[n1],int (*b)[n2],int m1,int n1,int n2,int (*c)[n2])
{

    for(int i = 0;i<m1;i++)
    {
        for(int j= 0;j<n2;j++)
        {
            c[i][j] = 0;
        }
}
for(int i=0;i<m1;i++)
    {
        for(int j=0;j<n2;j++)
        {
        int sum=0;
        for(int k=0;k<n1;k++)
        {
            sum=sum+a[i][k]*b[k][j];
            }
```

```
                    c[i][j]=sum;
                }
            }
        }
```

Let us write a program to take n names from the user and print it. Each name can have maximum of 20 characters.

```
#include<stdio.h>
 int main()
 {
        // Think about the commented code? Is it correct?
        /*
        char name1[20]; //Can store one name with maximum of 20 characters including '\0'
        // So need n different variables to store n names
        // But can we store n names under the same variable name???? --- Yes. Use 2D array
        */

        char names[200][20]; int n;
        printf("How many names you want to store?\n");
        scanf("%d", &n);
        int i;
	printf("enter %d names\n", n); for(i=0;i<n;i++)

	{

		scanf("%[^\n]s",names[i]);

	}

	printf("U entered below names\n"); for(int i=0;i<n;i++)

	{

		printf("%s\n",names[i]);

	}

	return 0;

}
```