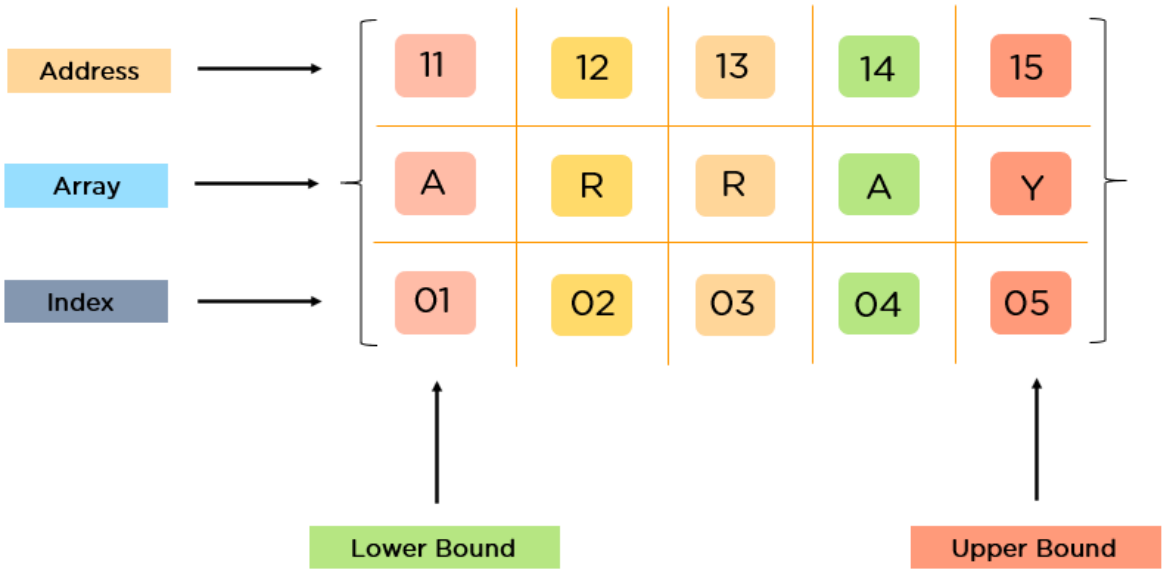
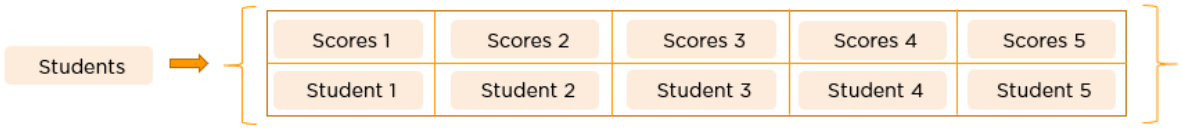
What Are Arrays in Data Structures?



An array is a linear data structure that collects elements of the same data type and stores them in contiguous and adjacent memory locations. Arrays work on an index system starting from 0 to (n-1), where n is the size of the array.

Why Do You Need an Array in Data Structures?

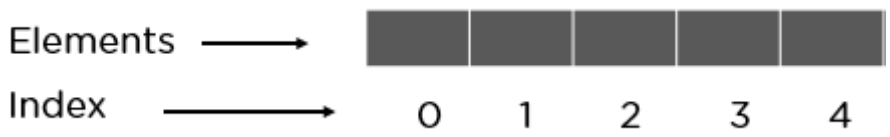


Let's suppose a class consists of ten students, and the class has to publish their results. If you had declared all ten variables individually, it would be challenging to manipulate and maintain the data. If more students were to join, it would become more difficult to declare all the variables and keep track of them. To overcome this problem, arrays came into the picture.

What Are the Types of Arrays?

There are majorly two types of arrays, they are:

One-Dimensional Arrays:

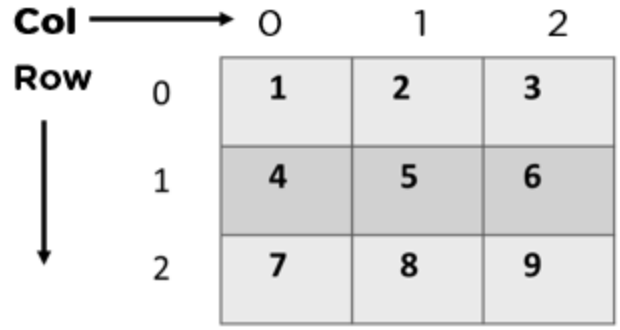


You can imagine a 1d array as a row, where elements are stored one after another.

Multi-Dimensional Arrays:

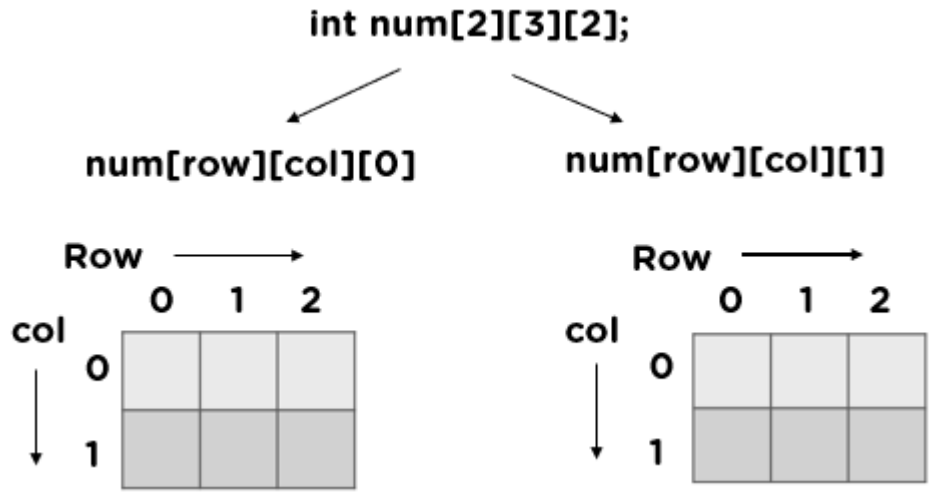
These multi-dimensional arrays are again of two types. They are:

Two-dimensional Arrays:

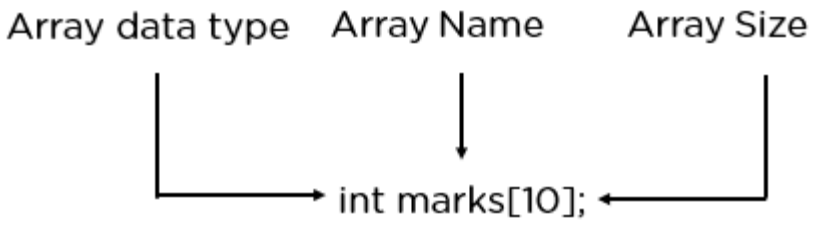


You can imagine it like a table where each cell contains elements.

Three-Dimensional Arrays:



You can imagine it like a cuboid made up of smaller cuboids where each cuboid can contain an element.



Arrays are typically defined with square brackets with the size of the arrays as its argument.

Here is the syntax for arrays:

1D Arrays: int arr[n];

2D Arrays: int arr[m][n];

3D Arrays: int arr[m][n][o];

How Do You Initialize an Array?

You can initialize an array in four different ways:

Method 1:

```
int a[6] = {2, 3, 5, 7, 11, 13};
```

Method 2:

```
int arr[]= {2, 3, 5, 7, 11};
```

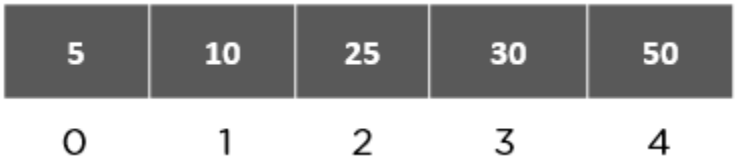
Method 3:

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

Method 4:

```
int arr[5];
arr[0]=1;
arr[1]=2;
arr[2]=3;
arr[3]=4;
arr[4]=5;
```

How Can You Access Elements of Arrays in Data Structures?



You can access elements with the help of the index at which you stored them. Let's discuss it with a code:

```
#include<stdio.h>

int main()

{

int a[5] = {2, 3, 5, 7, 11};

printf("%d\n",a[0]); // we are accessing

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

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

printf("%d\n",a[3]);

printf("%d",a[4]);

return 0;

}
```

```
2
3
5
7
11
--------------------------------
Process exited after 0.1476 seconds with return value 0
Press any key to continue . . .
```

What Operations Can You Perform on an Array?

- Traversal
- Insertion
- Deletion
- Searching
- Sorting

Traversing the Array:

Traversal in an array is a process of visiting each element once.

Code:

```
#include<stdio.h>
int main()
{
int a[5] = {2, 3, 5, 7, 11};
for(int i=0;i<5;i++)
{
//traversing ith element in the array
printf("%d\n",a[i]);
}
return 0;
}
```

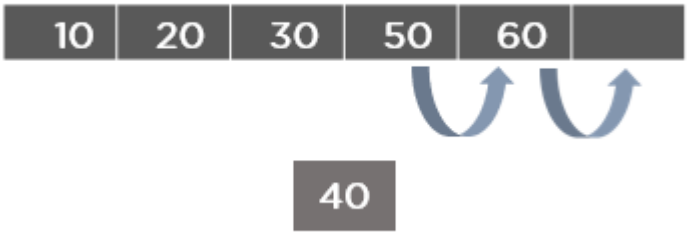
```
2
3
5
7
11
--------------------------------
Process exited after 0.1476 seconds with return value 0
Press any key to continue . . .
```

Insertion:

Insertion in an array is the process of including one or more elements in an array.

Insertion of an element can be done:

- At the beginning
- At the end and
- At any given index of an array.



At the Beginning:

Code:

```
#include<stdio.h>
int main()
{
   int array[10], n,i, item;
   printf("Enter the size of array: ");
    scanf("%d", &n);
   printf("\nEnter Elements in array: ");
      for(i=0;i<n;i++)
      {
         scanf("%d", &array[i]);
      }
   printf("\n enter the element at the beginning");
   scanf("%d", &item);
   n++;
   for(i=n; i>1; i--)
   {
         array[i-1]=array[i-2];
   }
   array[0]=item;
   printf("resultant array element");
   for(i=0;i<n;i++)
```

```
    {
        printf("\n%d", array[i]);
    }
    getch();
    return 0;
}
```

```
Enter the size of array: 5

Enter Elements in array: 2 3 5 7 11

enter the element at the beginning:1
resultant array element:
1
2
3
5
7
11
--------------------------------
Process exited after 19.94 seconds with return value 0
Press any key to continue . . .
```

At the End:

Code:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int array[10], i, values;
printf("Enter 5 Array Elements: ");
for(i=0; i<5; i++)
        scanf("%d", &array[i]);
printf("\nEnter Element to Insert: ");
scanf("%d", &values);
array[i] = values;
printf("\nThe New Array is:\n");
```

```
for(i=0; i<6; i++)
        printf("%d ", array[i]);
getch();
return 0;
}
```

```
Enter 5 Array Elements: 2 3 5 7 11

Enter Element to Insert: 13

The New Array is:
2 3 5 7 11 13
--------------------------------
Process exited after 16.76 seconds with return value 0
Press any key to continue . . .
```

At a Specific Position:

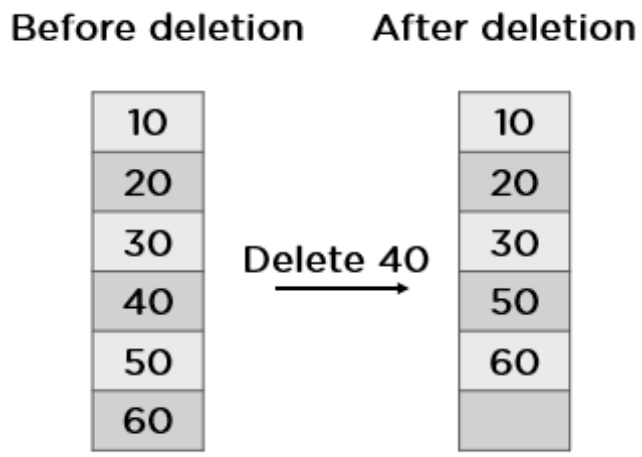
Code:

```
#include <stdio.h>
int main()
{
 int array[100], pos, size, val;
 printf("Enter size of the array:");
 scanf("%d", &size);
 printf("\nEnter %d elements\n", size);
 for (int i = 0; i < size; i++)
  scanf("%d", &array[i]);
 printf("Enter the insertion location\n");
 scanf("%d", &pos);
 printf("Enter the value to insert\n");
 scanf("%d", &val);
 for (int i = size - 1; i >= pos - 1; i--)
  array[i+1] = array[i];
 array[pos-1] = val;
```

```
 printf("Resultant array is\n");
 for (int i = 0; i <= size; i++)
  printf("%d\n", array[i]);
 return 0;
}
```

```
Enter size of the array:5

Enter 5 elements
2 3 5 7 11
Enter the insertion location
2
Enter the value to insert
13
Resultant array is
2
13
3
5
7
11

--------------------------------
Process exited after 14.82 seconds with return value 0
Press any key to continue . . .
```

Deletion:



Deletion of an element is the process of removing the desired element and re-organizing it.

You can also do deletion in different ways:

- At the beginning
- At the end

At the Beginning:

```
#include<stdio.h>
int main()
{
   int n,array[10];
   printf("enter the size of an array");
   scanf("%d" ,&n);
   printf("enter elements in an array");
   for(int i=0;i<n;i++)
      scanf("%d", &array[i]);
   n--;
   for(int i=0;i<n;i++)
    array[i]=array[i+1];
   printf("\nafter deletion ");
    for(int i=0;i<n;i++)
    printf("\n%d" , array[i]);
}
```

```
enter the size of an array: 5
enter elements in an array: 2 3 5 7 11

after deletion:
3
5
7
11

--------------------------------
Process exited after 7.997 seconds with return value 0
Press any key to continue . . .
```

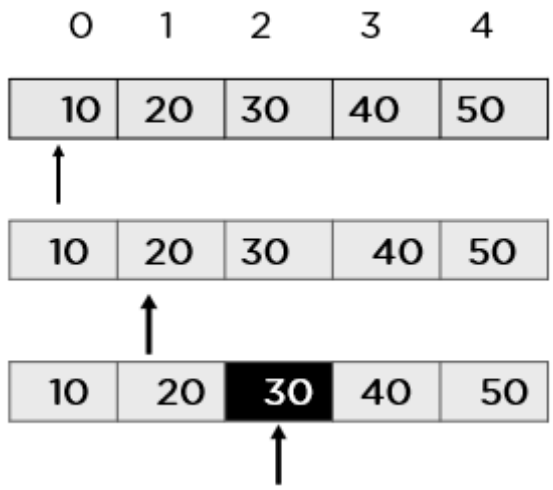
At the End:

```
#include<stdio.h>
int main()
{
int n,array[10];
```

```
printf("enter the size of an array");
scanf("%d" ,&n);
printf("enter elements in an array");
for(int i=0;i<n;i++)
scanf("%d", &array[i]);
printf("\nafter deletion array elements are");
for(int i=0;i<n-1;i++)
printf("\n%d" , array[i]);
}
```

```
enter the size of an array:5
enter elements in an array:2 3 5 7 11

after deletion array elements:
2
3
5
7

--------------------------------
Process exited after 15.99 seconds with return value 0
Press any key to continue . . .
```


Searching for an Element



The method of searching for a specific value in an array is known as searching.

There are two ways we can search in an array, they are:

- Linear search
- Binary search

Linear Search:

Code:

```
#include <stdio.h>
int linear(int a[], int n, int x)
{
for (int i = 0; i < n; i++)
if (a[i] == x)
return i;
return -1;
}
int main(void)
{
int a[] = { 2, 3, 4, 10, 40 };
int x = 10;
int n = sizeof(a) / sizeof(a[0]);
// Function call
int result = linear(a, n, x);
if(result == -1)
{
printf("Element is not present in array");
}
else
{
printf("Element found at index: %d", result);
}
return 0;
```

}

```
Element found at index: 3
--------------------------------
Process exited after 0.108 seconds with return value 0
Press any key to continue . . .
```

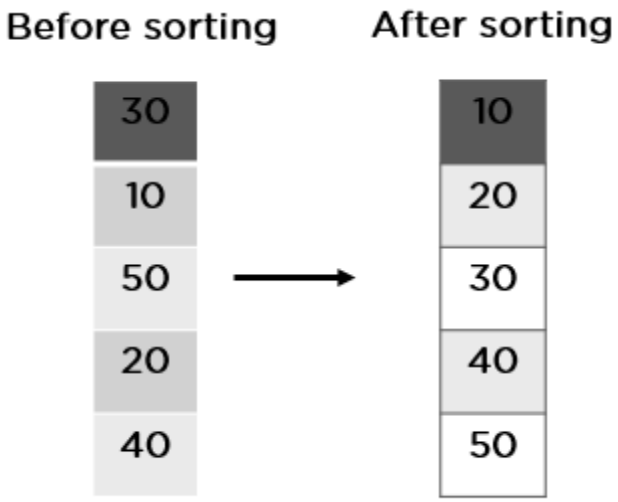
Binary Search:

```
#include <stdio.h>
int binary(int a[], int lt, int rt, int k)
{
if (rt >= lt) {
int mid = lt + (rt - l) / 2;
// check If element is at the middle
if (a[mid] == k)
return mid;
//check if element is at the left side of mid
if (a[mid] > x)
return binary(a, lt, mid - 1, k);
// Else element is at the right side of mid
return binary(a, mid + 1, rt, k);
}
// if element not found
return -1;
}
int main(void)
{
int a[] = { 2, 3, 5, 7, 11 };
int n = sizeof(a) / sizeof(a[0]);
int k = 11;
int res = binary(arr, 0, n - 1, k);
if(res == -1)
```

```
{
printf("Element is not found")
}
else
{
printf("Element found at index %d",res);
}
return 0;
}
```

```
Element found at index 4
--------------------------------
Process exited after 0.08186 seconds with return value 0
Press any key to continue . . .
```

Sorting:



Sorting in an array is the process in which it sorts elements in a user-defined order.

Code:

```
#include <stdio.h>
void main()
{
    int temp, size, array[100];
    printf("Enter the size \n");
```

```
    scanf("%d", &size);
    printf("Enter the numbers \n");
    for (int i = 0; i < size; ++i)
        scanf("%d", &array[i]);
    for (int i = 0; i < size; ++i)
    {
        for (int j = i + 1; j < size; ++j)
        {
        if (array[i] > array[j])
        {
                temp = array[i];
                array[i] = array[j];
                array[j] = temp;
        }
        }
    }
   printf("sorted array:\n");
    for (int i = 0; i < size; ++i)
        printf("%d\n", array[i]);
}
```

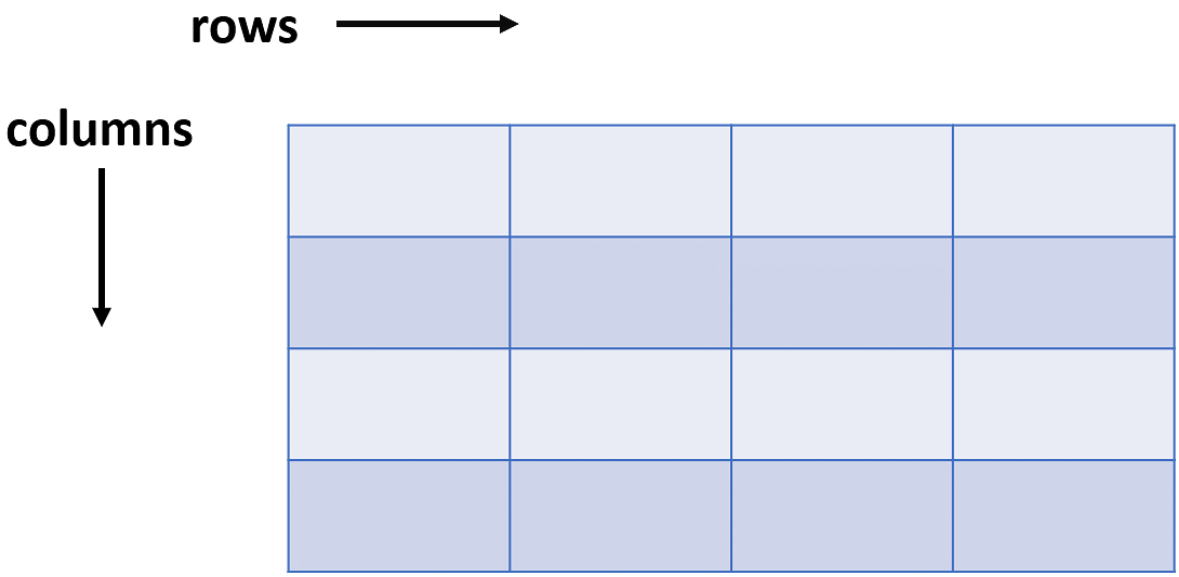
```
Enter the size
5
Enter the numbers
5 2 6 7 1
sorted array:
1
2
5
6
7

--------------------------------
Process exited after 34.49 seconds with return value 0
Press any key to continue . . .
```

What Are the Advantages of Arrays in Data Structures?

- Arrays store multiple elements of the same type with the same name.
- You can randomly access elements in the array using an index number.
- Array memory is predefined, so there is no extra memory loss.
- Arrays avoid memory overflow.
- 2D arrays can efficiently represent the tabular data.

What Are the Disadvantages of Arrays in Data Structures?

- The number of elements in an array should be predefined
- An array is static. It cannot alter its size after declaration.
- Insertion and deletion operation in an array is quite tricky as the array stores elements in continuous form.
- Allocating excess memory than required may lead to memory wastage.



The Need for Two-Dimensional Arrays:

Using 2d arrays, you can store so much data at one moment, which can be passed at any number of functions whenever required.

Picture this, a class consists of 5 students, and the class has to publish the result of all those students. You need a table to store all those five students' names, subjects' names, and marks. For that, it requires storing all information in a tabular form comprising rows and columns. A row contains the name of subjects, and columns contain the name of the students. That class consists of four subjects, namely English, Science, Mathematics, and Hindi, and the names of the students are first, second, third, fourth, and fifth.

	First	Second	Third	fourth	Fifth
English	**10**	**55**	**90**	**70**	**60**
Science	**50**	**60**	**75**	**65**	**95**
Mathematics	**95**	**75**	**55**	**85**	**45**
Hindi	**35**	**95**	**65**	**55**	**75**

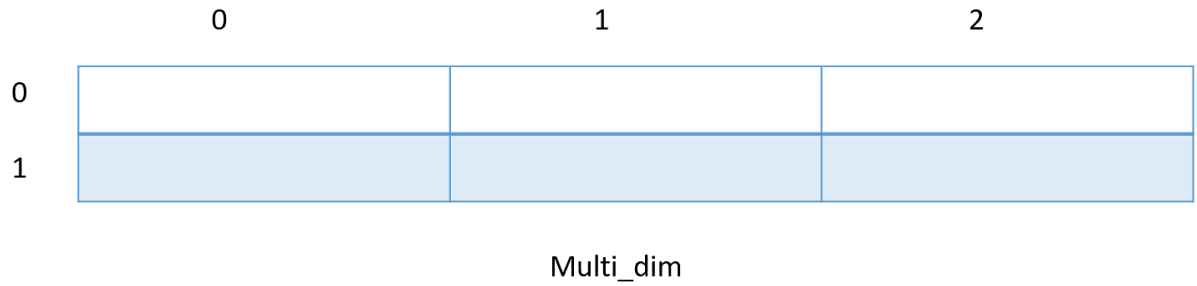
Declaration of Two-Dimensional Arrays:

The syntax of two-dimensional arrays is:

Data_type name_of_the_array[rows][index];

Here is one example:

int multi_dim[2][3];



Multi_dim

In the above example, the name of the 2d array is multi_dim consisting of 2 rows and three columns of integer data types.

Initialization of Two-Dimensional Arrays:

There are two methods to initialize two-dimensional arrays.

Method 1

int multi_dim[4][3]={10,20,30,40,50,60,20,80,90,100,110,120};

Method 2

int multi_dim[4][3]={{10,20,30},{40.50,60},{70,80,90},{100,110,120}};

Here are two methods of initialization of an element during declaration. Here, the second method is preferred because the second method is more readable and understandable so that you can clearly visualize that multi_dim 2D arrays comprise four rows and three columns.

Accessing Two-Dimensional Arrays:

Accessing two-dimensional arrays can be done using row index value and column index value.

Name_of_the arrays[row_index][column_index];

int multi_dim[4][3]={{10,20,30},{40,50,60},{70,80,90},{100,110,120}};

Suppose, in this example, you want to access element 80.

Multi_dim[2][1];

	0	1	2
0	**10**	**20**	**30**
1	**40**	**50**	**60**
2	**70**	**80**	**90**
3	**100**	**110**	**120**

Note: indexing always starts with zero.

Printing the Elements in a Two-Dimensional Array**.** Printing elements of a two-dimensional array can be done using two for loops.

```c

#include <stdio.h>

int main()
{
    int i,j;
    int multi_dim[4][3]={10,20,30,40,50,60,70,80,90,100,110,120};
    for(i=0;i<4;i++)
    {
        for(j=0;j<3;j++)
        {
            printf(" %d", multi_dim[i][j]);
        }
    }

    return 0;
}

```

```
input
10 20 30 40 50 60 70 80 90 100 110 120

...Program finished with exit code 0
Press ENTER to exit console.
```