Unit VII: Arrays and Strings (5)

Introduction, Dimension of Array, 1D Array Declaration, 1D Array Initialization, 1D Array input, 1D Array output

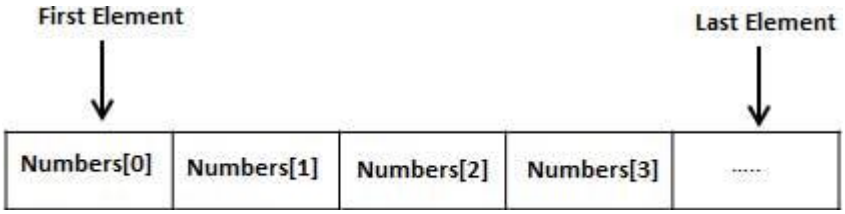
2D Array Declaration, 2D Array initialization, 2D Array input/output

String, String initialization String input/output, String Manipulation, 2D Array of String, Programming Examples

Introduction: Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Array variables are declared identically to variables of their data type, except that the variable name is followed by one pair of square [ ] brackets for each dimension of the array.

Dimension of Array: Arrays can be single or multidimensional. The number of subscript or index determines the dimensions of the array. An array of one dimension is known as a one-dimensional array or 1-D array, while an array of two dimensions is known as a two-dimensional array or 2-D array.

1. One-dimensional array or single dimension array: One dimensional array is a type of liner array in which only row index is present. It has single set of square bracket [ ].
   Syntax: datatype array name[size];
   datatype: It denotes the type of the elements in the array.

   array_name: Name of the array. It must be a valid identifier.
   size: Number of elements an array can hold. Here are some examples of array declarations:

   ```
   int num[100];
   float temp[20];
   char ch[50];
   ```

num is an array of type int, which can only store 100 elements of type int.
temp is an array of type float, which can only store 20 elements of type float.
ch is an array of type char, which can only store 50 elements of type char.

Example: An array named nums can be defined as

```
int nums[5];
```

This statement tells to the compiler that 'nums' is an array of type int and can store five integers. The compiler reserves 4 bytes of memory for each integer array element. Thus total memory size allocated by an integer array of size 5 is 20 bytes. It's individual elements are recognized by nums[0], nums[1], nums[2], nums[3] and nums[4]. The integer value within square bracket is called index of array. Index of an array always starts from 0 and ends with one less than size of array.

Initialization of 1-D array:

Till the array elements are not given any specific values, they are supposed to contain garbage values. The values can be assigned to elements at the time of array declaration, which is called array initialization. Since array has multiple elements, braces are used to denote the entire array and commas are used to separate the individual values assigned to the elements in the array initialization statements. The syntax for initialization is

```
data_type  array_name[size] = {value1, value2, .....valuen};
```

Where value1 is value of first element, value2 is that of second and so on.
The array initialization may be of three types as below.

a) `int a[5] = {1,2,3,5,8};`

Here, a is integer type array which has 5 elements. Their values are assigned as 1, 2, 3, 4, 5, 8 (i.e a[0] = 1, a[1] = 2, a[2] = 3, a[3] = 5, a[4] = 8). The array elements are stored sequentially in separate memory locations.

b) `int b[ ]= {2,5,7};`

Here, size of array is not given, the compiler automatically sets its size according to the number of values given. The size of array b is 3in this array initialization as three values 2, 5 and 7 are assigned at the time of initialization. Thus, writing `int b[ ]= {2, 5, 7};` is equivalent to writing `int b[3] = {2, 5, 7};`

c) `int c[10] = {45,89, 54, 8, 9};`

In this example, the size of array c has been set 10 but only 5 elements are assigned at the time of initialization. In this situation, all individual elements that are not assigned explicitly contains zero as initial values. Thus, the value of c[5], c[6], c[7], c[8], and c[9] is zero in this example.

Accessing array elements: The particular array element in an array is accessed by specifying the name of array, followed by square bracket enclosing an integer, which is called the array index. The array index indicates the particular element of the array which we want to access. The numbering of elements starts from zero and ends to a number one less than the size of the array. For example if an array marks is defined as

float marks[5];

Then, the array elements are marks[0], marks[1], marks [2], marks[3] and marks[4]. We can assign float values to these individual elements directly. i.e.

```
marks[0] = 90.7;
marks[1] = 45;
marks[2] = 78.5;
marks[3] = 0;
marks[4] = 45.8;
```

The loop can be used to input and output the elements of array.

Example:

Write a program that reads 10 integers from keyboard, stores in an array and displays entered numbers in the screen.

```
#include<stdio.h>
int main()
{
int a[10], i;
printf("Enter 10 numbers:\t");
for(i=0;i<10;i++)
scanf("%d", &a[i]);
printf("\n you have entered these 10 numbers:\n");
for(i=0;i<10;i++)
printf("\t a[%d] =%d", i, a[i]);
return 0;
        }
```

Example:

Write a program to illustrate the memory locations allocated by each array elements.

```
#include<stdio.h>
int main()
{
float a[]= { 10.4,45.9,5.5,0,10.6};
int i;
printf("The continuous memory locations are:\n");
for(i=0;i<5;i++)
```

```
printf("\t%d", &a[i]);
return 0;
}
```

Example: Write a program to sort n numbers in ascending order.

```
#include<stdio.h>
int main()
{
int nums[50], i, j, n, temp;
printf("\n How many numbers are there?\t");
scanf("%d", &n);
printf("\n Enter %d numbers:\n", n);
for(i=0;i<n;i++)
scanf("%d",&nums[i]);
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(nums[i]>nums[j])
{
temp=nums[i];
nums[i]= nums[j];
nums[j]  = temp;
}
}
}
printf("\n The numbers in ascending order:\n");
for(i=0;i<n;i++)
printf("\t%d", nums[i]);
return 0;
}
```

Multi-Dimensional Arrays: Multidimensional arrays are those which have more than one dimension. Multi- dimensional arrays are defined in much the same manner as one dimensional array, except that a separate pair of square brackets is required for each subscript or dimension or index. Thus, two dimensional arrays will require two pairs of square brackets. The two dimensional array is also called matrix. A m*n two dimensional array can be thought as tables of values having m rows and n columns. For example int X[3][3] can be shown as follows

	Col 1	**Col 2**	**Col 3**
Row0	X[0][0]	X[0][1]	X[0][2]
Row1	X[1][0]	X[1][1]	X[1][2]
Row2	X[2][0]	X[2][1]	X[2][2]

Declaration of two-dimensional array: Like one dimensional array, two dimensional arrays must also be declared before using it. The syntax for declaration is

data_type array_name[row_size][col_size];

Example:

int matrix[2][3]; /* matrix is 2-D array which has 2 rows and 3 columns i.e it contains 6 elements*/

float m[10][20];

char students[10][15];

Initialization of multi-dimensional array : Multi-dimensional arrays can be initialized at the time of array definition. Some examples of 2-D array initialization are:

(i) **int marks[2][3] =** { (2, 4, 6), (8, 10, 12) };

is equivalent to following assignments.

marrks[0][0] =2; marrks[0][1] =4; marrks[0][2] =6;

marks[1][0] =8; marks[1][1] =10; marks[1][2] = 12;

(ii) int marks

1. **Program for 2D array input and output**

```
#include <stdio.h>
int main()
{
int row,col;
inti,j;
int arr[50][50];
printf("enter the number of rows:");
scanf("%d",&row);
printf("enter the of columns:");
scanf("%d",&col);
for(i=0;i<row;i++)
   {
for(j=0;j<col;j++)
         {
printf("enter the value for arr[%d][%d]:",i,j);
scanf("%d",&arr[i][j]);
         }
   }
for(i=0;i<row;i++)
   {
for(j=0;j<col;j++)
      {
printf("\t%d",arr[i][j]);
      }
printf("\n");
   }
return 0;}
```

2. Addition two matrix

```
#include <stdio.h>
int main()
{
int r1,c1,r2,c2;
printf("\n enter number of rows of first matrix\n");
scanf("%d", &r1);
printf("\n enter the columns of first matrix\n");
scanf("%d",&c1);
printf("\n enter number of rows of second matrix\n");
scanf("%d", &r2);
printf("\n enter the columns of second matrix\n");
scanf("%d",&c2);
if(r1==r2&&c1==c2)
   {
int i, j;
int A[r1][c1];
int B[r2][c2];
int C[r2][c2];
printf("\n Enter the elements of first matrix\n");
for(i=0;i<r1;i++)
      {
for(j=0;j<c1;j++)
{
scanf("%d",&A[i][j]);
}
}
printf("\n Enter second matrix\n");
for(i=0;i<r2;i++)
{
for(j=0;j<c2;j++)
{
scanf("%d",&B[i][j]);
 }
 }
for(i=0;i<r1;i++)
 {
for(j=0;j<c1;j++)
 {
      C[i][j]=A[i][j]+B[i][j];
 }
}
printf("\n resultant matrix\n");
for(i=0;i<r1;i++)
      {
for(j=0;j<c1;j++)
```

```
      {
printf("\t%d",C[i][j]);
      }
printf("\n");  } }
else {
printf("The addition of matrix is not possible");    }
return 0;}
```