UNIT – III

12

CHAPTER

ARRAYS

12.1 INTRODUCTION

Definition

An array is a derived data type. It is a collection of data elements of similar data type under a common name. Each element in the collection is identified by an index. The data elements are stored in a contiguous memory location. C supports rich set of both primitive and user derived data types. Array is one such example of a derived data type. Like basic primitive data type variables, array variables are also used for arithmetic and logical expressions.

General syntax for array:

datatype array_name[n]

Ex int a[10];

int is the data type of the elements

a is the array name

n is **array_size - 10**

where, 10 is the index which denotes the array containing 10 data elements.

Now an array **a** can hold 10 integer elements.

Characteristics of an Array:

1. All the data elements share the same name and they are distinguished from one another with the help of an index number.
2. Any particular element of an array can be modified separately without disturbing other elements.
3. The index number in an array plays major role for calling each element.
4. Any particular element of an array can be independently used for calculation.

Example

```
c=2.4*a[5];
if(a[3]>a[8])
x=(p>m)?a[2]:a[1];
```

12.2 CLASSIFICATION OF ARRAY

Arrays can be classified into three major types:

1. One dimensional Array
2. Two dimensional Array
3. Multidimensional Array

12.3 ONE DIMENSIONAL ARRAY DECLARATION

Like normal variables, an array variable also needs to be declared before its first usage. The memory allocation to an array is done at the compile time.

Declaration of an array:

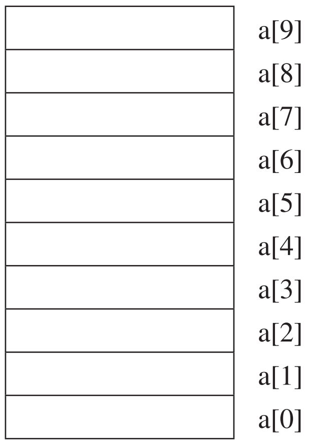
```
        int a[10];
        float x[10];
        char s[15];
```

int a[10] declares **a** as an array of 10 integers. Each integer needs two byte memory for its storage. So totally a block of 20 byte memory will be allocated for this array.

float x[10] declares × as an array of 10 real numbers. Each real needs 4 byte memory for its storage. So totally a block of 40 byte memory will be allocated for this array.

Char s[15] declares **s** as an array of 15 characters. Each character needs 1 byte memory for its storage. So totally a block of 15 byte memory will be allocated for this array.

Individual elements of an array are stored in a contiguous memory location as follows:



12.4 ONE DIMENSIONAL ARRAY INITIALIZATION

Like normal variables, an array can be initialized with static data at compile time or dynamic data at run time. Static declaration always assigns constant values to array elements during declaration. Dynamic declaration assigns different values to array elements during run time for each execution.

Example

```
int a[5]={1,2,3,4,5}  // static declaration
```

Always assign these five values only to all the five elements of an array. Here, five integers are stored in the array called **a**. The array elements are stored sequentially in separate locations. Individual array elements are referred as below:

a[0] refers to element 1
a[1] refers to element 2
a[2] refers to element 3
a[3] refers to element 4
a[4] refers to element 5

If we represent array name **a** without any index, it indirectly specifies the 1st element a[0]. The array index starts from 0 to n-1 size. Therefore 1st element is referred by a[0], 2nd element is referred by a[1], and the last nth element is referred by a[n-1]. If we try to access the array index beyond the boundary, the system will assign garbage value for that element and proceeds further. It won't generate any error message. Such situation will generate unexpected results. So care must be ensured that we should access the array index within the allowable limit.

One dimensional array elements are represented by an array name followed by a single index, for example, a[10]. The array elements are arranged in rows or columns. The data are stored in continuous memory location. One dimensional array is used to handle a set of values and their manipulations.

Example

Element	a[0]	a[1]	a[2]	a[3]	a[4]
Address	2000	2002	2004	2006	2008

Normally one dimensional array means a set of values. In order to read values for an array or printing the array elements in specific format, we need a looping structure. Either do..while, while or for loop is used.

Program segment to read five integers to an array 'A' as follows:

Using while loop

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

Using do while loop

```
     int A[5]
     int n=0;
     do
     {
          scanf( "%d", &A[n]);
          n++;
     } while(n<5);
```

Using for loop

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

12.5 SAMPLE PROGRAMS

12.5.1 Program to read 10 elements in an array and print the same

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10],n,i;
clrscr();
printf("\nEnter the size of the array");
scanf("%d",&n);  // n must be less than 10
printf("\nEnter the elements into the array one by one");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("\nThe array elements are\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
getch();
}
```

OUTPUT

```
Enter the size of the array5

Enter the elements into the array one by one
1
```

```
2
3
4
5

The array elements are
1       2       3       4       5
```

12.5.2 Program to sort 10 integers in an array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10],n,i,j,t;
clrscr();
printf("\nEnter the size of the array…");
        scanf("%d",&n);
printf("\nEnter the elements into the array one by one…");
for(i=0;i<n;i++)  // n must be less 10
        scanf("%d",&a[i]);
for(i=0;i<n;i++)
{
        for(j=i+1;j<n;j++)
        {
              if( a[i] <= a[j] )
              {
                 t    = a[i];
                 a[i] = a[j];
                 a[j] = t;
              }
        }
}
printf("\nThe array elements are…\n");
for(i=0;i<n;i++)
        printf("%d\t",a[i]);
printf("\nThe array elements in reverse order are…\n");
for(i=n-1;i<=0;i--)
        printf("%d\t",a[i]);
getch();
}
```

OUTPUT

```
Enter the size of the array….5

Enter the elements into the array one by one…
11
```

```
6
22
15
5

The array elements are…
5      6      11     15     22
The array elements in reverse order are….
22     15     11      6      5
```

12.5.3 Program to search an element present in the array or not

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10],n,i,x;
int FLAG=0;
clrscr();
printf("\nEnter the size of the array …");
        scanf("%d",&n);
printf("\nEnter the elements into the array one by one …");
for(i=0;i<n;i++)  // n must be less 10
        scanf("%d",&a[i]);
printf("\n Enter an element to be searched in the array…");
scanf("%d",&x);
for(i=0;i<n;i++)
{
        if( a[i] == x )
           {
           FLAG = 1;
           printf("\n The element %d present in the array at %d position",x,i);
           }
        if(FLAG == 1)
           printf("\n The element %d not present in the array",x);
}
getch();
}
```

OUTPUT

```
Enter the size of the array….
5

Enter the elements into the array one by one…
33
66
99
```

```
88
22

Enter an element to be searched …
88

The element 88 is present in the array at the 4 position.
```

Another run

```
Enter the size of the array….
5
Enter the elements into the array one by one…
45
12
8
97
22
Enter an element to be searched … 71
The element 71 is not present in the array
```

12.5.4 Program to find the standard deviation of n values

```
#include<stdio.h>
#include<conio.h>
#define SIZE 100
Void main()
{
int a[SIZE],n,i,mean;
float sum,sumsqr,stddeviation,variance;
sum = sumsqr=0.0;
printf("\n Enter the size of input array…");
scanf("%d",&n);
printf("\n Enter the  input values     …");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
sum = sum + a[i];
}
mean = sum / n;
for(i=0;i<n;i++)
{
      stddeviation = a[i] - mean;
      sumsqr = sumsqr + (stddeviation * stddeviation);
}
variance = sumsqr / n;
stddeviation = sqrt(variance);
printf("\n  Numbers of items … %d",n);
printf("\n Mean….    %d",mean);
printf("\\n Standard Deviation   = %f ",stddeviation);
}
```

Output

```
Enter the size of input array
8
Enter the input values…
65
9
27
78
12
20
33
49

Number of items … 8
Mean = 36.62500
Standard Deviation = 23.510303
```

12.5.5 Program to find sum of the elements in an array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[25],i,j,n,sum=0;
clrscr();
printf("\nEnter the no. of elements in the array");
scanf("%d",&n);
printf("\nEnter the elements in the array\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=0;i<n;i++)
{
sum=sum+a[i];
}
printf("\nThe sum of the elements in the array is %d",sum);
getch();
}
```

OUTPUT

```
Enter the number of elements in the array4
Enter the elements in the array
4
3
2
1
The sum of the elements in the array is 10
```

12.5.6 Program to insert an element into an array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[30],i,n,x,p;
clrscr();
printf("\nEnter the size of the array");
scanf("%d",&n);
printf("\nenter the elements into the array");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("\nEnter the element to be inserted");
scanf("%d",&x);
printf("\nEnter the position in which the element to be inserted");
scanf("%d",&p);
for(i=n-1;i>=p;i--)
a[i+1]=a[i];
a[p]=x;
printf("\nThe elements are \n");
for(i=0;i<=n;i++)
printf("%d\t",a[i]);
getch();
}
```

OUTPUT

```
Enter the size of the array  5
Enter the elements into the array
3       4       5       2       1
Enter the element to be inserted  5
Enter the position in which the element to be inserted  3
The elements are:
3       4       5       5       2       1
```

12.5.7 Program to find the largest element in an array

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[25],i,n,max;
clrscr();
printf("\nEnter the no. of elements in the array");
scanf("%d",&n);
printf("\nEnter the elements in the array\n");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
max=a[0];
for(i=1;i<n;i++)
{
if(a[i]>max)
max=a[i];
}
printf("\nThe largest element is %d",max);
getch();
}
```

OUTPUT

```
Enter the number of elements in the array4

Enter the elements in the array
6
7
5
45

The largest element is 45
```

12.6 TWO-DIMENSIONAL ARRAY

Two dimensional array variables are used to handle a table of values. The array elements are stored in the form of rows and columns. It can be represented by an array name followed by two indices. First index is used to represent the row position and the second index represents the column position in the table.

General syntax for a two dimensional array:

data_type array_name[n][m];
n – represents row size
m – represents column size

12.7 TWO DIMENSIONAL ARRAY DECLARATION

int a[3][2], b[3][3];

Here a[3][2] means array **a** contains 3 rows and 2 columns. So in total (3*2) = 6 data are given in the table. Similarly array **b** contains 3 rows and 3 columns. In total 9 data are given in the table. Two dimensional arrays are used to handle matrix elements conveniently.

Diagrammatic representation for array a[3][2]

	Column 1	**Column 2**
Row 1	a[0][0]	a[0][1]
Row 2	a[1][0]	a[1][1]
Row 3	a[2][0]	a[2][1]

Diagrammatic representation for array b[3][3]

	Column 1	**Column 2**	**Column 3**
Row 1	b[0][0]	b[0][1]	b[0][0]
Row 2	b[1][0]	b[1][1]	b[1][0]
Row 3	b[2][0]	b[2][1]	b[2][0]

The above arrangement of array elements is only for understanding. Physically array elements are stored in continuous memory locations. The two dimensional array is a collection of one-dimensional array, which are placed one after another. First row elements are stored continuously according to row major order.

12.8 TWO DIMENSIONAL ARRAY INITIALIZATION

Array can be initialized either statically at compile time or dynamically at run time.

Compile time initialization

It always assigns constant values to array elements at declaration itself._

Example:

```
int a[2][3] = { (2,4,6),(1,3,5)};
int b[3][3] = { (1,1,1),(2,2,2),(3,3,3)};
```

Run time initialization

Dynamic values are initialized during run time. In order to process two dimensional array elements, we need nested looping structure.

Example: Program segment to read a table or a matrix of size 2 * 3

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

The control variables i controls the row size and j controls the column size.

Limitations of an Array:

1. The memory of an array is allocated at the compile time.
2. Arrays are static in nature. The size of array is fixed. We cannot change the size according to the requirement.
3. The size of an array has to be kept long enough to accommodate the worst cases. Therefore, memory usage in case of arrays is inefficient.

12.9 PROGRAMS IN TWO DIMENSIONAL ARRAY

12.9.1 Matrix Addition and Subtraction

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10][10],b[10][10],c[10][10],d[10[10],i,j,k,m,n;
clrscr();
printf("\nEnter the no. of rows in the matrix A");
scanf("%d",&m);
printf("\nEnter the no. of columns in the matrix A");
scanf("%d",&n);
printf("\nEnter the elements in the matrix A");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\nEnter the elements in the matrix B");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
```

```
scanf("%d",&b[i][j]);
}
}
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
}
printf("\nThe Resultant addition matrix is\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}

for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
d[i][j]=a[i][j]-b[i][j];
}
}
printf("\nThe Resultant subtracted matrix is\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",d[i][j]);
}
printf("\n");
}
getch();
}
```

OUTPUT

```
2
2
2
2
Enter the number of rows in the matrix2
Enter the number of columns in the matrix2
Enter the elements in the matrix A
1
1
1
1
Enter the elements in the matrix B
1
1
1
1
The Resultant addition matrix is
3       3
3       3
The Resultant subtracted matrix is
1       1
1       1
```

12.9.2 Matrix Multiplication

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10][10],b[10][10],c[10][10],i,j,k,m,n,p;
clrscr();
printf("\nEnter the no. of rows in the matrix A");
scanf("%d",&m);
printf("\nEnter the no. of columns in the matrix A");
scanf("%d",&n);
printf("\nEnter the no. of columns in the matrix B");
scanf("%d",&p);
if( n !=0)
printf("\n Matrix multiplication is not possible")
else
{
printf("\nEnter the elements in the matrix A");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\nEnter the elements in the matrix B");
for(i=0;i<n;i++)
{
for(j=0;j<p;j++)
{
scanf("%d",&b[i][j]);
}
}
printf("\nThe Resultant matrix is\n");
for(i=0;i<m;i++)
{
for(j=0;j<p;j++)
{
c[i][j]=0;
for(k=0;k<n;k++)
{
c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
printf("%d\t",c[i][j]);
}
printf("\n");
}
getch();
}
}
```

OUTPUT

```
Enter the number of rows in the matrix A 2
Enter the number of columns in the matrix A 2
Enter the number of columns in the matrix B 2
Enter the elements in the matrix A
1
1
1
1
Enter the elements in the matrix B
1
1
1
1
The Resultant matrix is
2       2
2       2
```

12.9.3 Sum of Diagonal Elements in a Matrix

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10][10],i,j,m,n,sum=0;
clrscr();
printf("\nEnter the no. of rows in the matrix A");
scanf("%d",&m);
printf("\nEnter the no. of columns in the matrix A");
scanf("%d",&n);
printf("\nEnter the elements in the matrix A");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\nThe Addition of diagonal elements in the matrix is\n");
for(i=0;i<m;i++)
{
sum=sum+a[i][i];
}
printf("%d",sum);
getch();
}
```

OUTPUT

```
Enter the number of rows in the matrix A2
Enter the number of columns in the matrix A2
Enter the elements in the matrix A
1
1
1
1
The addition of diagonal elements in the matrix is
2
```

12.10 MULTIDIMENSIONAL ARRAY

C compiler supports three or more dimensions. Exact limit is decided by the compiler. However, most compilers permit seven to ten dimensions.

The general syntax of a multidimensional array is follows:

```
data_type  array_name[s1][s2]….s[n];
```

where, s_i – size of the i^{th} dimension.

Example

```
int table]2][3]4]
```

Table given below is a three dimensional array variable containing (2*3*4) 24 elements of integer data. First dimension three represents table number, second dimension represents row number in the particular table and the third dimension represents the column number in the same table.

Table 1

	c0	c1	c2	c3
r0				
r1				
r2				

Table 2

	c0	c1	c2	c3
r0				
r1				
r2				

12.11 MORE ABOUT ARRAYS

In this chapter we have discussed the basic concepts of arrays and their applications to some extent. Arrays can be effectively utilized along with pointers and memory management functions. Some repetitive operations using arrays can be carried out with the help of functions, structures. These aspects of arrays are covered in the later chapters.

12.11.1 Character Array

The data elements stored in the array belong to character data type. Hence the name, character array.

The character array can be used to store individual characters or a collection of characters called strings. If a string is stored in the character array, a null character ('\0') is automatically added to the array at the end, which denotes the end of the string.

One dimensional character array is enough to represent a set of characters in a single string. But we need a two dimensional array to represent a set names or strings.

Example

```
char  S[10];
char  P[5][10];
```

S[10] is a character array which is able to hold 10 characters to represent a single string. S[0] represent 1st character, S[1] represent 2nd character and so on. P[5][10] is an array of 5 strings, each having a maximum of 10 characters. P[0] represent 1st string, P[4] represent fifth string. P[0][5] represent 5th character in 1st string.

EXAMPLE PROGRAM

Program to display character array & their addresses

```
#include<stdio.h>
void main()
{
char a[]={'A','B','C','D'};
int i;
clrscr();
for(i=0;a[i]!='\0';i++)
{
printf("\nThe array elements are : %c",a[i]);
printf("\nThe memory location is %u",&a[i]);
}
getch();
}
```

OUTPUT

```
The array elements are: A
The memory location is 65522
The array elements are: B
The memory location is 65523
The array elements are: C
The memory location is 65524
The array elements are: D
The memory location is 65525
```