UNIT 6 ARRAYS

Structure

6.0 INTRODUCTION

C language provides four basic data types - *int, char, float and double.* We have learnt about them in Unit 3. These basic data types are very useful; but they can handle only a limited amount of data. As programs become larger and more complicated, it becomes increasingly difficult to manage the data. Variable names typically become longer to ensure their uniqueness. And, the number of variable names makes it difficult for the programmer to concentrate on the more important task of correct coding. Arrays provide a mechanism for declaring and accessing several data items with only one identifier, thereby simplifying the task of data management.

Many programs require the processing of multiple, related data items that have common characteristics like *list* of numbers, marks in a course, or enrolment numbers. This could be done by creating several individual variables. But this is a hard and tedious process. For example, suppose you want to read in five numbers and print them out in reverse order. You could do it the hard way as:

```
main()
{
 int al,a2,a3,a4,a5;
 scanf("%d %d %d %d %d",&a1,&a2,&a3,&a4,&a5);
 printf("%d %d %d %d %d",a5,a4,a3,a2,a1);
}
```

Does it look good if the problem is to read in 100 or more related data items and print them in reverse order? Of course, the solution is the use of the regular variable names **a1**, **a2** and so on. But to remember each and every variable and perform the operations on the variables is not only tedious a job and disadvantageous too. One common organizing technique is to use arrays in such situations. An array is a collection of similar kind of data elements stored in adjacent memory locations and are referred to by a single array-name. In the case of C, you have to declare and define **array** before it can be used. Declaration and definition tell the compiler the name of the array, the type of each element, and the size or number of elements.To explain it, let us consider to store marks of five students. They can be stored using five variables as follows:

```
int ar1, ar2, ar3, ar4, ar5;
```

Now, if we want to do the same thing for 100 students in a class then one will find it difficult to handle 100 variables. This can be obtained by using an array. An array declaration uses its size in [ ] brackets. For above example, we can define an array as:

int ar [100];

where *ar* is defined as an array of size 100 to store marks of integer data-type. Each element of this collection is called an *array-element* and an integer value called the *subscript* is used to denote individual elements of the array. An *ar* array is the collection of 200 consecutive memory locations referred as below:

Figure 6.1: Representation of an array

In the above figure, as each integer value occupies 2 bytes, 200 bytes were allocated in the memory.

This unit explains the use of arrays, types of arrays, declaration and initialization with the help of examples.

6.1 OBJECTIVES

After going through this unit you will be able to:

- declare and use arrays of one dimension;
- initialize arrays;
- use subscripts to access individual array elements;
- write programs involving arrays;
- do searching and sorting; and
- handle multi-dimensional arrays.

6.2 ARRAY DECLARATION

Before discussing how to declare an array, first of all let us look at the characteristic features of an array.

- Array is a data structure storing a group of elements, all of which are of the same data type.
- All the elements of an array share the same name, and they are distinguished from one another with the help of an index.
- Random access to every element using a numeric index (subscript).
- A simple data structure, used for decades, which is extremely useful.
- Abstract Data type (ADT) *list* is frequently associated with the array data structure.

The declaration of an array is just like any variable declaration with additional *size* part, indicating the number of elements of the array. Like other variables, arrays must be declared at the beginning of a function.

The declaration specifies the base type of the array, its name, and its size or dimension. In the following section we will see how an array is declared:

6.2.1 Syntax of Array Declaration

Syntax of array declaration is as follows:

data-type array_name [constant-size];

Data-type refers to the type of elements you want to store
Constant-size is the number of elements

The following are some of declarations for arrays:

```
int      char [80];
float    farr [500];
static   int iarr [80];
char     charray [40];
```

There are two restrictions for using arrays in C:

- The amount of storage for a declared array has to be specified at **compile time** before execution. This means that an array has a fixed size.
- The data type of an array applies uniformly to all the elements; for this reason, an array is called a **homogeneous** data structure.

6.2.2 Size Specification

The size of an array should be declared using symbolic constant rather a fixed integer quantity (The subscript used for the individual element is of are integer quantity). The use of a symbolic constant makes it easier to modify a program that uses an array. All reference to maximize the array size can be altered simply by changing the value of the symbolic constant. (Please refer to Unit – 3 for details regarding symbolic constants).

To declare size as 50 use the following symbolic constant, SIZE, defined:

#define SIZE 50

The following example shows how to declare and read values in an array to store marks of the students of a class.

Example 6.1

Write a program to declare and read values in an array and display them.

```
/* Program to read values in an array*/

#  include < stdio.h >
#  define  SIZE   5                          /* SIZE is a symbolic constant */

main ( )
{
int  i = 0;                                  /* Loop variable */
int   stud_marks[SIZE]; /* array declaration */

/* enter the values of the elements */
for( i = 0;i<SIZE;i++)
    {
       printf  (“Element no. =%d”,i+1);
       printf(“ Enter the value of the element:”);
```

```
        scanf("%d",&stud_marks[i]);
    }
printf("\nFollowing are the values stored in the corresponding array elements: \n\n");
for( i = 0; i<SIZE;i++)
        {
            printf("Value stored in a[%d] is %d\n"i, stud_marks[i]);
        }
}
```

OUTPUT:

Element no. = 1 Enter the value of the element = 11
Element no. = 2 Enter the value of the element = 12
Element no. = 3 Enter the value of the element = 13
Element no. = 4 Enter the value of the element = 14
Element no. = 5 Enter the value of the element = 15

Following are the values stored in the corresponding array elements:

Value stored in a[0] is 11
Value stored in a[1] is 12
Value stored in a[2] is 13
Value stored in a[3] is 14
Value stored in a[4] is 15

6.3 ARRAY INITIALIZATION

Arrays can be initialized at the time of declaration. The initial values must appear in the order in which they will be assigned to the individual array elements, enclosed within the braces and separated by commas. In the following section, we see how this can be done.

6.3.1 Initialization of Array Elements in the Declaration

The values are assigned to individual array elements enclosed within the braces and separated by comma. Syntax of array initialization is as follows:

data type array-name [size] = {val 1, val 2, ...val n};

val 1 is the value for the first array element, *val 2* is the value for the second element, and *val n* is the value for the *n* array element. Note that when you are initializing the values at the time of declaration, then there is no need to specify the size. Let us see some of the examples given below:

```
int digits  [10] = {1,2,3,4,5,6,7,8,9,10};

int digits[ ] = {1,2,3,4,5,6,7,8,9,10};

int vector[5] = {12,-2,33,21,13};

float temperature[10] ={ 31.2, 22.3, 41.4, 33.2, 23.3, 32.3, 41.1, 10.8, 11.3, 42.3};

double width[ ] = { 17.33333456, -1.212121213, 222.191345 };

int height[ 10 ] = { 60, 70, 68, 72, 68 };
```

6.3.2 Character Array Initialisation

The array of characters is implemented as strings in C. Strings are handled differently as far as initialization is concerned. A special character called null character ***' \0 '***, implicitly suffixes every string. When the external or static string character array is assigned a string constant, the size specification is usually omitted and is automatically assigned; it will include the '\0'character, added at end. For example, consider the following two assignment statements:

char thing [3] = "TIN";
char thing [ ] = "TIN";

In the above two statements the assignments are done differently. The first statement is not a string but simply an array storing three characters 'T', 'I' and 'N' and is same as writing:

char thing [3] = {'T', 'I', 'N'};

whereas, the second one is a four character string TIN\0. The change in the first assignment, as given below, can make it a string.

char thing [4] = "TIN";

Check Your Progress 1

1. What happens if I use a subscript on an array that is larger than the number of elements in the array?
   ……
   ……
2. Give sizes of following arrays.
   a. char carray [ ]= "HELLO";
   b. char carray [5]= "HELLO";
   c. char carray [ ]={ 'H', 'E', 'L', 'L', 'O' };
   ……
   ……
3. What happens if an array is used without initializing it?
   ……
   ……
4. Is there an easy way to initialize an entire array at once?
   ……
   ……
5. Use a *for* loop to total the contents of an integer array called numbers with five elements. Store the result in an integer called TOTAL.
   ……
   ……

6.4 SUBSCRIPT

To refer to the individual element in an array, a subscript is used. Refer to the statement we used in the Example 6.1,

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

Subscript is an integer type constant or variable name whose value ranges from 0 to SIZE - 1 where SIZE is the total number of elements in the array. Let us now see how we can refer to individual elements of an array of size 5:

Consider the following declarations:

```
char country[ ] = "India";
int stud[ ] = {1, 2, 3, 4, 5};
```

Here both arrays are of size 5. This is because the country is a char array and initialized by a string constant "India" and every string constant is terminated by a null character '\0'. And stud is an integer array. country array occupies 5 bytes of memory space whereas stud occupies size of 10 bytes of memory space. The following table: 6.1 shows how individual array elements of *country* and *stud* arrays can be referred:

Table 6.1: Reference of individual elements

Element no.	Subscript	country array		stud array	
		Reference	Value	Reference	Value
1	0	country [0]	'I'	stud [0]	1
2	1	country [1]	'n'	stud [1]	2
3	2	country [2]	'd'	stud [2]	3
4	3	country [3]	'i'	stud [3]	4
5	4	country [4]	'a'	stud [4]	5

Example 6.2

Write a program to illustrate how the marks of 10 students are read in an array and then used to find the maximum marks obtained by a student in the class.

```
/* Program to find the maximum marks among the marks of 10 students*/

#  include < stdio.h >
#  define  SIZE   10                          /* SIZE is a symbolic constant */

main ( )
{

int  i = 0;
int max = 0;
int   stud_marks[SIZE]; /* array declaration */

/* enter the values of the elements */
for( i = 0;i<SIZE;i++)
    {
      printf  ("Student no. =%d",i+1);
      printf(" Enter the marks out of 50:");
      scanf("%d",&stud_marks[i]);
    }

/* find maximum */
for (i=0;i<SIZE;i ++)
    {
     if (stud_marks[i]>max)
      max = stud_marks[ i ];
     }
```

```
printf("\n\nThe maximum of the marks obtained  among all the 10 students is: %d
      ",max);
}
```

OUTPUT

```
Student no. = 1   Enter the marks out of 50: 10
Student no. = 2   Enter the marks out of 50: 17
Student no. = 3   Enter the marks out of 50: 23
Student no. = 4   Enter the marks out of 50: 40
Student no. = 5   Enter the marks out of 50: 49
Student no. = 6   Enter the marks out of 50: 34
Student no. = 7   Enter the marks out of 50: 37
Student no. = 8   Enter the marks out of 50: 16
Student no. = 9   Enter the marks out of 50: 08
Student no. = 10 Enter the marks out of 50: 37

The maximum of the marks obtained among all the 10 students is: 49
```

6.5 PROCESSING THE ARRAYS

For certain applications the assignment of initial values to elements of an array is required. This means that the array be defined globally (extern) or locally as a static array.

Let us now see in the following example how the marks in two subjects, stored in two different arrays, can be added to give another array and display the average marks in the below example.

Example 6.3:

Write a program to display the average marks of each student, given the marks in 2 subjects for 3 students.

```
/* Program to display the average marks of 3 students */

# include < stdio.h >
# define SIZE 3
main()
{
int  i  = 0;
float   stud_marks1[SIZE];        /* subject 1array declaration */
float   stud_marks2[SIZE];        /*subject 2 array declaration */
float   total_marks[SIZE];
float   avg[SIZE];

printf("\n Enter the marks in subject-1 out of 50 marks: \n");
for( i = 0;i<SIZE;i++)
              {
                printf("Student no. =%d",i+1);
                printf(" Enter the marks= ");
                scanf("%f",&stud_marks1[i]);
              }
printf("\n Enter the marks in subject-2 out of 50 marks \n");
   for(i=0;i<SIZE;i++)
              {
```

```
            printf("Student no. =%d",i+1);
            printf(" Please enter the marks= ");
           scanf("%f",&stud_marks2[i]);
          }

    for(i=0;i<SIZE;i++)
    {
          total_marks[i]=stud_marks1[i]+ stud_marks2[i];
                avg[i]=total_marks[i]/2;
                printf("Student no.=%d, Average= %f\n",i+1, avg[i]);
        }
        }
```

OUTPUT

```
Enter the marks in subject-1out of 50 marks:
Student no. = 1  Enter the marks= 23
Student no. = 2  Enter the marks= 35
Student no. = 3  Enter the marks= 42

Enter the marks in subject-2 out of 50 marks:
Student no. = 1  Enter the marks= 31
Student no. = 2  Enter the marks= 35
Student no. = 3  Enter the marks= 40

Student no. = 1 Average= 27.000000
Student no. = 2 Average= 35.000000
Student no. = 3 Average= 41.000000
```

Let us now write another program to search an element using the linear search.

Example 6.4

Write a program to search an element in a given list of elements using Linear Search.

```
/* Linear Search.*/

# include<stdio.h>
# define SIZE   05
main()
{
int  i  = 0;
int  j;
int   num_list[SIZE];      /* array declaration */

/* enter elements in the following loop */

printf("Enter any 5 numbers: \n");
for(i = 0;i<SIZE;i ++)
    {
        printf("Element no.=%d Value of the element=",i+1);
        scanf("%d",&num_list[i]);
    }
printf ("Enter the element to be searched:");
scanf ("%d",&j);

/* search using linear search */
for(i=0;i<SIZE;i++)
```

```
    {
    if(j == num_list[i])
          {
       printf("The number exists in the list at position: %d\n",i+1);
            break;
          }
    }
}
```

OUTPUT

```
Enter any 5 numbers:
Element no.=1 Value of the element=23
Element no.=2 Value of the element=43
Element no.=3 Value of the element=12
Element no.=4 Value of the element=8
Element no.=5 Value of the element=5
Enter the element to be searched: 8
The number exists in the list at position: 4
```

Example 6.5

Write a program to sort a list of elements using the selection sort method

```
/* Sorting list of numbers using selection sort method*/

#include <stdio.h>
#define SIZE 5

main()
{

int j,min_pos,tmp;
int i;                          /* Loop variable */
int a[SIZE];      /* array declaration */

/* enter the elements  */

for(i=0;i<SIZE;i++)
   {
    printf("Element no.=%d",i+1);
    printf("Value of the element: ");
    scanf("%d",&a[i]);
    }

/* Sorting by descending order*/

for (i=0;i<SIZE;i++)
  {
   min_pos = i;
   for (j=i+1;j<SIZE;j++)
     if (a[j] < a[min_pos])
            min_pos = j;
   tmp = a[i];
   a[i] = a[min_pos];
   a[min_pos] = tmp;
 }
```

```
/* print the result */

printf("The array after sorting:\n");
      for(i=0;i<SIZE;i++)
        printf("% d\n",a[i]);
}
```

OUTPUT

```
Element no. = 1 Value of the element: 23
Element no. =2  Value of the element: 11
Element no. = 3 Value of the element: 100
Element no. = 4 Value of the element: 42
Element no. = 5 Value of the element: 50

The array after sorting:
11
23
42
50
100
```

Check Your Progress 2

1. Name the technique used to pass an array to a function.
   ……

   ……
2. Is it possible to pass the whole array to a function?
   ……

   ……
3. List any two applications of arrays.
   ……

   ……

6.6 MULTI-DIMENSIONAL ARRAYS

Suppose that you are writing a chess-playing program. A chessboard is an 8-by-8 grid. What data structure would you use to represent it? You could use an array that has a chessboard-like structure, i.e. a *two-dimensional array*, to store the positions of the chess pieces. Two-dimensional arrays use two indices to pinpoint an individual element of the array. This is very similar to what is called "algebraic notation", commonly used in chess circles to record games and chess problems.

In principle, there is no limit to the number of subscripts (or dimensions) an array can have. Arrays with more than one dimension are called *multi- dimensional arrays*. While humans cannot easily visualize objects with more than three dimensions, representing multi-dimensional arrays presents no problem to computers. In practice, however, the amount of memory in a computer tends to place limits on the size of an array . A simple four-dimensional array of double-precision numbers, merely twenty elements wide in each dimension, takes up 20^4 * 8, or 1,280,000 bytes of memory - about a megabyte.

For exmaple, you have ten rows and ten columns, for a total of 100 elements. It's really no big deal. The first number in brackets is the number of rows, the second number in brackets is the number of columns. So, the upper left corner of any grid

would be element [0][0]. The element to its right would be [0][1], and so on. Here is a little illustration to help.

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]
[2][0]	[2][1]	[2][2]

Three-dimensional arrays (and higher) are stored in the same way as the two-dimensional ones. They are kept in computer memory as a linear sequence of variables, and the last index is always the one that varies fastest (then the next-to-last, and so on).

6.6.1 Multi - Dimensional Array Declaration

You can declare an array of two dimensions as follows:

datatype array_name[*size1*][*size2*];

In the above example, *variable_type* is the name of some type of variable, such as int. Also, *size1* and *size2* are the sizes of the array's first and second dimensions, respectively. Here is an example of defining an 8-by-8 array of integers, similar to a chessboard. Remember, because C arrays are zero-based, the indices on each side of the chessboard array run 0 through 7, rather than 1 through 8. The effect is the same: a two-dimensional array of 64 elements.

int chessboard [8][8];

To pinpoint an element in this grid, simply supply the indices in both dimensions.

6.6.2 Initialisation of Two - Dimensional Arrays

If you have an *m* x *n* array, it will have *m * n* elements and will require *m*n*element-size* bytes of storage. To allocate storage for an array you must reserve this amount of memory. The elements of a two-dimensional array are stored row wise. If table is declared as:

int table [2] [3] = { 1,2,3,4,5,6 };

It means that element
table [0][0] = 1;
table [0][1] = 2;
table [0][2] = 3;
table [1][0] = 4;
table [1][1] = 5;
table [1][2] = 6;

The neutral order in which the initial values are assigned can be altered by including the groups in { } inside main enclosing brackets, like the following initialization as above:

```
int     table  [ 2 ] [ 3 ] = { {1,2,3},
                               {4,5,6}    };
```

The value within innermost braces will be assigned to those array elements whose last subscript changes most rapidly. If there are few remaining values in the row, they will be assigned zeros. The number of values cannot exceed the defined row size.

```
int     table [ 2 ] [ 3 ] = { { 1, 2, 3},{ 4}};
```

It assigns values as
table [0][0] = 1;
table [0][1] = 2;
table [0][2] = 3;
table [1][0] = 4;
table [1][1] = 0;
table [1][2] = 0

Remember that, C language performs no error checking on array bounds. If you define an array with 50 elements and you attempt to access element 50 (the 51st element), or any out of bounds index, the compiler issues no warnings. It is the programmer's task to check that all attempts to access or write to arrays are done only at valid array indexes. Writing or reading past the end of arrays is a common programming bug and is hard to isolate.

Check Your Progress 3

1. Declare a multi-dimensioned array of floats called balances having three rows and five columns.
   ……
   ……
2. Write a *for* loop to total the contents of the multi-dimensioned float array balances.
   ……
   ……
3. Write a for loop which will read five characters (use scanf) and deposit them into the character based array words, beginning at element 0.
   ……
   ……

6.7 SUMMARY

Like other languages, C uses arrays as a way of describing a collection of variables with identical properties. The group has a single name for all its members, with the individual member being selected by an *index*. We have learnt in this unit, the basic purpose of using an array in the program, declaration of array and assigning values to the arrays. All elements of the arrays are stored in the consecutive memory locations**.** Without exception, all arrays in C are indexed from 0 up to one less than the bound given in the declaration. This is very puzzling for a beginner. Watch out for it in the examples provided in this unit. One important point about array declarations is that they don't permit the use of varying subscripts. The numbers given must be constant expressions which can be evaluated at compile time, not run time. As with other variables, global and static array elements are initialized to 0 by default, and automatic array elements are filled with garbage values. In C, an array of type char is used to represent a character string, the end of which is marked by a byte set to 0 (also known as a NULL character).

Whenever the arrays are passed to function their starting address is used to access rest of the elements. This is called – Call by reference. Whatever changes are made to the

elements of an array in the function, they are also made available in the calling part. The formal argument contains no size specification except for the rightmost dimension. Arrays and pointers are closely linked in C. Multi-dimensional arrays are simply arrays of arrays. To use arrays effectively it is a good idea to know how to use pointers with them. More about the pointers can be learnt from Unit -10 (Block -3).

6.8 SOLUTIONS / ANSWERS

Check Your Progress 1

1. If you use a subscript that is out of bounds of the array declaration, the program will probably compile and even run. However, the results of such a mistake can be unpredictable. This can be a difficult error to find once it starts causing problems. So, make sure you're careful when initializing and accessing the array elements.

2.
   a) 6
   b) 5
   c) 5

3. This mistake doesn't produce a compiler error. If you don't initialize an array, there can be any value in the array elements. You might get unpredictable results. You should always initialize the variables and the arrays so that you know their content.

4. Each element of an array must be initialized. The safest way for a beginner is to initialize an array, either with a declaration, as shown in this chapter, or with a ***for*** statement. There are other ways to initialize an array, but they are beyond the scope of this Unit.

5. Use a ***for*** loop to total the contents of an integer array which has five elements. Store the result in an integer called total.

```
for ( loop = 0, total = 0; loop < 5; loop++ )
total = total + numbers[loop];
```

Check Your Progress 2

1. Call by reference.

2. It is possible to pass the whole array to a function. In this case, only the address of the array will be passed. When this happens, the function can change the value of the elements in the array.

3. Two common statistical applications that use arrays are:

   - **Frequency distributions**: A frequency array show the number of elements with an identical value found in a series of numbers. For example, suppose we have taken a sample of 50 values ranging from 0 to 10. We want to know how many of the values are 0, how many are 1, how many are 2 and so forth up to 10. Using the arrays we can solve the problem easily . Histogram is a pictorial representation of the frequency array. Instead of printing the values of the elements to show the frequency of each number, we print a histogram in the form of a bar chart.
   - **Random Number Permutations**: It is a set of random numbers in which no numbers are repeated. For example, given a random number permutation of 5 numbers, the values of 0 to 5 would all be included with no duplicates.

Check Your Progress 3

1. float balances[3][5];

2. for(row = 0, total = 0; row < 3; row++)
   for(column = 0; column < 5; column++)
   total = total + balances[row][column];

3. for(loop = 0; loop < 5; loop++)
   scanf ("%c", &words[loop]);

6.9 FURTHER READINGS

1. The C Programming Language, *Brain W. Kernighan, Dennis M. Ritchie*, PHI.
2. C, The Complete Reference, Fourth Edition, *Herbert Schildt*, TMGH, 2002.
3. Computer Science – A Structured Programming Approach Using C, *Behrouz A. Forouzan, Richard F. Gilberg*, Thomas Learning, Second edition, 2001.
4. Programming with ANSI and TURBO C, *Ashok N. Kamthane*, Pearson Education, 2002.