UNIT 4 ARRAYS AND STRINGS

Structure

4.0 INTRODUCTION

C language provides four basic data types - *int, char, float and double.* We have learnt about them in Unit 2. 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:

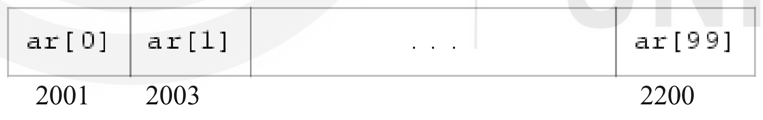

Figure 4.1: Representation of an Array

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 in the first few sections and later on focuses on string handling in C programming language.

4.1 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;
- handle multi-dimensional arrays;
- define, declare and initialize a string;
- discuss various formatting techniques to display the strings; and
- discuss various built-in string functions and their use in manipulation of strings.

4.2 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:

4.2.1 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.

4.2.2 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 – 2 for details regarding symbolic constants).

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

#defineSIZE   50

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

Example 4.1

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
```

4.3 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.

4.3.1 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 };
```

4.3.2 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.

   ……………………………………………………………………………………
   ……………………………………………………………………………………
   ……………………………………………………………………………………
   ……………………………………………………………………………………

4.4 SUBSCRIPT

To refer to the individual element in an array,a subscript is used. Refer to the statement we used in the Example 4.1,

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: 4.1 shows how individual array elements of *country* and *stud* arrays can be referred:

Table 4.1: 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 4.2

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
```

4.5 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 4.3

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("\nEnter 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 4.4

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 4.5

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.

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

4.6 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 example, 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).

4.6.1 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.

4.6.2 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.

   …………………………………………………………………………
   …………………………………………………………………………
   …………………………………………………………………………
   …………………………………………………………………………
   …………………………………………………………………………

4.7 INTRODUCTION TO STRINGS

In the previous unit, we have discussed numeric arrays, a powerful data storage method that lets you group a number of same-type data items under the same group name. Individual items, or elements, in an array are identified using a subscript after the array name. Computer programming tasks that involve repetitive data processing lend themselves to array storage. Like non-array variables, arrays must be declared before they can be used. Optionally, array elements can be initialized when the array is declared. In the earlier unit, we had just known the concept of *character arrays* which are also called *strings*.

String can be represented as a single-dimensional character type array. C language does not provide the intrinsic string types. Some problems require that the characters within a string be processed individually. However, there are many problems which require that strings be processed as complete entities. Such problems can be manipulated considerably through the use of special string oriented library functions. Most of the C compilers include string library functions that allow string comparison, string copy, concatenation of strings etc. The string functions operate on null-terminated arrays of characters and require the header <string.h>.The use of the some of the string library functions are given as examples in this unit.

4.8 DECLARATION AND INITIALIZATION OF STRINGS

Strings in C are group of characters, digits, and symbols enclosed in quotation marks or simply we can say the string is declared as a "character array". The end of the string is marked with a special character, the '\0' (*Null character)*, which has the decimal value 0. There is a difference between a *character* stored in memory and a *single character string* stored in a memory. The character requires only one byte whereas the single character string requires two bytes (one byte for the character and other byte for the delimiter).

Declaration of Strings

A string in C is simply a sequence of characters. To declare a string, specify the data type as char and place the number of characters in the array in square brackets after the string name. The syntax is shown as below:

char string-name[size];

For example,
char name[20];
char address[25];
char city[15];

Initialization of Strings

The string can be initialized as follows:

char name[8]={'P','R','O','G','R','A','M','\0'};

Each character of string occupies 1 byte of memory (on 16 bit computing). The size of character is machine dependent, and varies from 16 bit computers to 64 bit computers. The characters of strings are stored in the contiguous (adjacent) memory locations.

1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
P	R	O	G	R	A	M	\0
1001	1002	1003	1004	1005	1006	1007	1008

The C compiler inserts the NULL (\0) character automatically at the end of the string. So initialization of the NULL character is not essential.

You can set the initial value of a character array when you declare it by specifying a string literal. If the array is too small for the literal, the literal will be truncated. If the literal (including its null terminator) is smaller than the array, then the final characters in the array will be undefined. If you don't specify the size of the array, but do specify a literal, then C will set the array to the size of the literal, including the null terminator.

```
  char str[4]={‘u’, ‘n’, ‘i’, ‘x’};
  char str[5]={‘u’, ‘n’, ‘i’, ‘x’, ‘\0’};
  char str[3];
  char str[]=“UNIX”;
  char str[4]=“unix”;
  char str[9]=“unix”;
```

All of the above declarations are legal. But which ones don’t work? The first one is a valid declaration, but will cause major problems because it is not *null-terminated.* The second example shows a correct null-terminated string. The special escape character **\0** denotes string termination. The fifth example suffers the size problem, the character array ***‘str’*** is of size 4 bytes, but it requires an additional space to store ***‘\0’***. The fourth example however does not. This is because the compiler will determine the length of the string and automatically initialize the last character to a null-terminator. The strings not terminated by a ***‘\0’*** are merely a collection of characters and are called as *character arrays*.

String Constants

String constants have double quote marks around them, and can be assigned to char pointers. Alternatively, you can assign a string constant to a char array - either with no size specified, or you can specify a size, but don’t forget to leave a space for the null character! Suppose you create the following two code fragments and run them:

```
/*   Fragment 1   */
{
   char *s;
   s=hello”;
   printf(“%s\n”,s);
}

/*  Fragment  2 */
{
   char s[100];
   strcpy(s,“ hello”);
   printf(“%s\n”,s);
}
```

These two fragments produce the same output, but their internal behaviour is quite different. In fragment 2, you cannot say **s="hello";**. To understand the differences, you have to understand how the *string constant table* works in C. When your program is compiled, the compiler forms the object code file, which contains your machine code and a table of all the string constants declared in the program. In fragment 1, the statement **s="hello";** causes ***s*** to point to the address of the string **hello** in the string constant table. Since this string is in the string constant table, and therefore technically a part of the executable code, you cannot modify it. You can only point to it and use it in a read-only manner. In fragment 2, the string **hello** also exists in the constant table, so you can copy it into the array of characters named ***s***. Since ***s*** is not an address, the statement **s="hello";** will not work in fragment 2. It will not even compile.

Example 4.6

Write a program to read a name from the keyboard and display message **Hello** onto the monitor".

```
/*Program that reads the name and display the hello along with your name*/
#include <stdio.h>
main()
{
char name[10];
printf("\nEnter Your Name :);
scanf("%s", name);
printf("Hello %s\n",name);
}
```

OUTPUT

```
Enter Your Name:  Alex
Hello Alex
```

In the above example declaration char name [10] allocates 10 bytes of memory space(on 16 bit computing) to array name []. We are passing the base address to scanf function and scanf() function fills the characters typed at the keyboard into array until enter is pressed. The scanf() places '\0' into array at the end of the input. The printf() function prints the characters from the array on to monitor, leaving the end of the string '\0'. The *%s* used in the scanf() and printf() functions is a format specification for strings.

4.9 DISPLAY OF STRINGS USING DIFFERENT FORMATTING TECHNIQUES

The ***printf*** function with ***%s*** format is used to display the strings on the screen. For example, the below statement displays entire string:

printf("%s", name);

We can also specify the accuracy with which character array (string) is displayed. For example, if you want to display first 5 characters from a field width of 15 characters, you have to write as:

printf("%15.5s", name);

If you include minus sign in the format (e.g. % –10.5s), the string will be printed left justified.

printf("% -10.5s", name);

Example 4.7

Write a program to display the string "UNIX" in the following format.

```
U
UN
UNI
UNIX
UNIX
UNI
```

```
          UN
          U
/* Program to display the string in the above shown format*/

# include<stdio.h>
main()
{
int  x, y;
static char string[]="UNIX";
printf("\n");
for( x=0; x<4; x++)
{
          y=x + 1;
   /* reserves 4 character of space on to the monitor and minus sign is for left
justified*/
          printf("%-4.*s \n", y, string);

  /* and for every loop the * is replaced by value of y */
/* y value starts with 1 and for every time it is incremented by 1 until it reaches to 4*/
}

for( x=3; x>=0; x- -)
          {
           y=x +1;
           printf("%-4.*s \n", y, string);
/*  y value starts with 4 and for every time it is decrements by 1 until it reaches to 1*/
       }
}
```

OUTPUT

```
U
UN
UNI
UNIX
UNIX
UNI
UN
U
```

4.10 ARRAY OF STRINGS

Array of strings are multiple strings, stored in the form of table. Declaring array of strings is same as strings, except it will have additional dimension to store the number of strings. Syntax is as follows:

char array-name[size][size];

For example,

char names[5][10];

where names is the name of the character array and the constant in first square brackets will gives number of string we are going to store, and the value in second square bracket will gives the maximum length of the string.

Example 4.8

char names [3][10]={"martin","phil","collins"};

It can be represented by a two-dimensional array of size[3][10] as shown below:

0	1	2	3	4	5	6	7	8	9
m	a	r	t	i	n	\0			
p	h	i	l	\0					
c	o	l	l	i	n	s	\0		

Example 4.9

Write a program to initializes 3 names in an array of strings and display them on to monitor

/* Program that initializes 3 names in an array of strings and display them on to monitor.*/

```
#include <stdio.h>
main()
{
        int n;
        char names[3][10]={“Alex”, “Phillip”, “Collins” };
        for(n=0; n<3; n++)
        printf(“%s \n”,names[n] );    }
```

OUTPUT

```
Alex
Phillip
Collins
```

Check Your Progress 4

1. Which of the following is a static string?

   A. Static String;

   B. “Static String”;

   C. ‘Static String’;

   D. char string[100];

   …………………………………………………………………………………

   …………………………………………………………………………………

   …………………………………………………………………………………

2. Which character ends all strings?

   A. ‘.’
   B. ‘ ‘

   C. ‘0’

   D. ‘n’

   …………………………………………………………………………………

   …………………………………………………………………………………

   …………………………………………………………………………………

3. What is the Output of the following programs?

(a)
```
main()
{
    char name[10]="IGNOU";
    printf("\n %c", name[0]);
    printf("\n %s", name);
}
```

(b)
```
main()
{
  char s[]="hello";
  int j=0;
  while( s[j] !='\0' )
      printf(" %c",s[j++]);
}
```

(c)
```
main()
{
  char str[]="hello";
  printf("%10.2s", str);
  printf("%-10.2s", str);
}
```

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

4 Write a program to read 'n' number of lines from the keyboard using a two-dimensional character array (ie., strings).

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

4.11 BUILT IN STRING FUNCTIONS AND APPLICATIONS

The header file <string.h> contains some string manipulation functions. The following is a list of the common string managing functions in C.

4.11.1 Strlen Function

The **strlen** function returns the length of a string. It takes the string name as argument. The syntax is as follows:

n=strlen(str);

where **str** is name of the string and **n** is the length of the string, returned by **strlen** function.

Example 4.10

Write a program to read a string from the keyboard and to display the length of the string on to the monitor by using strlen( ) function.

```
/*  Program to illustrate the strlen function to determine the length of a string */

#include <stdio.h>
#include <string.h>
main()
{
char name[80];
int length;
printf("Enter your name: ");
gets(name);
length=strlen(name);
printf("Your name has %d characters\n", length);
}
```

OUTPUT

```
Enter your name: TYRAN
Your name has 5 characters
```

4.11.2 Strcpy Function

In C, you cannot simply assign one character array to another. You have to copy element by element. The string library <string.h> contains a function called **strcpy** for this purpose. The **strcpy** function is used to copy one string to another. The syntax is as follows:

strcpy(str1, str2);
where str1, str2 are two strings. The content of string str2 is copied on to string str1**.**

Example 4.11

Write a program to read a string from the keyboard and copy the string onto the second string and display the strings on to the monitor by using strcpy( ) function.

```
/* Program to illustrate strcpy function*/

#include <stdio.h>
#include <string.h>
main()
{
char first[80], second[80];
printf("Enter a string: ");
gets(first);
strcpy(second, first);
printf("\n First string is : %s, and second string is: %s\n", first, second);
}
```

OUTPUT

```
Enter a string: ADAMS
First string is: ADAMS, and second string is: ADAMS
```

4.11.3 Strcmp Function

The **strcmp** function in the string library function which compares two strings, character by character and stops comparison when there is a difference in the ASCII value or the end of any one string and returns ASCII difference of the characters that is integer. If the return value ***zero*** means the two strings are equal, a negative value means that first is less than second, and a positive value means first is greater than second**.** The syntax is as follows:

n=strcmp(str1, str2);

where **str1** and **str2** are two strings to be compared and **n** is returned value of differed characters.

Example 4.12

Write a program to compare two strings using string compare function.

```
/* The following program uses the strcmp function to compare two strings. */

#include <stdio.h>
#include <string.h>
main()
{
 char first[80], second[80];
 int value;
printf("Enter a string: ");
 gets(first);
 printf("Enter another string: ");
 gets(second);
 value=strcmp(first,second);
  if(value ==0)
        puts("The two strings are equal");
     else if(value < 0)
          puts("The first string is smaller ");
         else if(value > 0)
              puts("the first string is bigger");
}
```

OUTPUT

```
Enter a string: MOND
Enter another string: MOHANT
The first string is smaller
```

4.11.4 Strcat Function

The **strcat** function is used to join one string to another. It takes two strings as arguments; the characters of the second string will be appended to the first string. The syntax is as follows:

strcat(str1, str2);

where str1 and str2 are two string arguments, string *str2* is appended to string *str1***.**

Example 14.13

Write a program to read two strings and append the second string to the first string.

```
/* Program for string concatenation*/

#include <stdio.h>
#include <string.h>
main()
{
char first[80], second[80];
printf("Enter a string:");
gets(first);
printf("Enter another string: ");
gets(second);
strcat(first, second);
printf("\nThe two strings joined together: %s\n", first);
}
```

OUTPUT

```
Enter a string: BOREX
Enter another string: BANKS
The two strings joined together: BOREX BANKS
```

4.11.5 Strlwr Function

The **strlwr** function converts upper case characters of string to lower case characters. The syntax is as follows:

strlwr(str1);
where str1 is string to be converted into lower case characters.

Example 4.14

Write a program to convert the string into lower case characters using in-built function.

```
/* Program that converts input string to lower case characters */

#include <stdio.h>
#include <string.h>
main()
{
char first[80];
printf("Enter a string: ");
gets(first);
printf("Lower case of the string is %s", strlwr(first));
}
```

OUTPUT

```
Enter a string: BROOKES
Lower case of the string is brookes
```

4.11.6 Strrev Function

The **strrev** funtion reverses the given string. The syntax is as follows:

strrev(str);
where string **str** will be reversed.

Example 4.15

Write a program to reverse a given string.

```
/* Program to reverse a given string */

#include <stdio.h>
#include <string.h>
main()
{
char first[80];
printf("Enter a string:");
gets(first);
printf("\n Reverse of the given string is :  %s ", strrev(first));
}
```

OUTPUT

```
Enter a string: ADANY
Reverse of the given string is:  YNADA
```

4.11.7 Strspn Function

The **strspn** function returns the position of the string, where first string mismatches with second string. The syntax is as follows:

n=strspn(first, second);

where **first** and **second** are two strings to be compared, **n** is the number of character from which first string does not match with second string.

Example 4.16

Write a program, which returns the position of the string from where first string does not match with second string.

```
/*Program which returns the position of the string from where first string does
not match with second string*/

#include <stdio.h>
#include <string.h>
main()
{
char first[80], second[80];
printf("Enter  first string: ");
gets(first);
printf("\n Enter  second string: ");
gets(second);
printf("\n After %d characters there is no match",strspn(first, second));
}
```

OUTPUT

```
Enter first string: ALEXANDER
Enter second string: ALEXSMITH
After 4 characters there is no match
```

4.11.8 Other String Functions

strncpy function

The **strncpy** function same as *strcpy*. It copies characters of one string to another string up to the specified length. The syntax is as follows:

strncpy(str1, str2, 10);
where str1 and str2 are two strings. The **10** characters of string **str2** are copied onto string **str1.**

stricmp function

The **stricmp** function is same as *strcmp*, except it compares two strings ignoring the case(lower and upper case). The syntax is as follows:

n=stricmp(str1, str2);

strncmp function

The **strncmp** function is same as *strcmp*, except it compares two strings up to a specified length. The syntax is as follows:

n=strncmp(str1, str2, 10);
where **10** characters of **str1** and **str2** are compared and **n** is returned value of differed characters.

strchr function

The **strchr** funtion takes two arguments(the string and the character whose address is to be specified) and returns the address of first occurrence of the character in the given string. The syntax is as follows:

cp=strchr(str, c);
where **str** is string and **c** is character and **cp** is character pointer.

strset function

The **strset** funtion replaces the string with the given character**.** It takes two arguments the string and the character. The syntax is as follows:

strset(first, ch);
where string **first** will be replaced by character **ch**.

strchr function

The **strchr** function takes two arguments(the string and the character whose address is to be specified) and returns the address of first occurrence of the character in the given string. The syntax is as follows:

cp=strchr(str, c);
where **str** is string and **c** is character and **cp** is character pointer.

strncat function

The **strncat** function is the same as *strcat*, except that it appends upto specified length. The syntax is as follows:

strncat(str1, str2,10);
where 10 character of the str2 string is added into str1 string.

strupr function

The strupr function converts lower case characters of the string to upper case characters. The syntax is as follows:

strupr(str1);
where str1 is string to be converted into upper case characters.

strstr function

The **strstr** function takes two arguments address of the string and second string as inputs. And returns the address from where the second string starts in the first string. The syntax is as follows:

cp=strstr(first, second);
where **first** and s**econd** are two strings, **cp** is character pointer.

Check Your Progress 5

1. Which of the following functions compares two strings?

A. compare();
B. stringcompare();
C. cmp();
D. strcmp();

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

2. Which of the following appends one string to the end of another?
A. append();

B. stringadd();

C. strcat();

D. stradd();

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

3. Write a program to concatenate two strings without using the *strcat()* function.

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

4. Write a program to find string length without using the *strlen()* function.

   ……………………………………………………………………………….

   ……………………………………………………………………………….

   ……………………………………………………………………………….

5. Write a program to convert lower case letters to upper case letters in a given string without using strupp().

   ……………………………………………………………………………….

   ……………………………………………………………………………….

   ……………………………………………………………………………….

4.12 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 and also the string handling functions. 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 -7 (Block -2).

Strings are sequence of characters. Strings are to be null-terminated if you want to use them properly. Remember to take into account null-terminators when using dynamic memory allocation. The string.h library has many useful functions. Losing the ‘ **\0’** character can lead to some very considerable bugs. Make sure you copy \0 when you copy strings. If you create a new string, make sure you put \0 in it. And if you copy one string to another, make sure the receiving string is big enough to hold the source string, including \0. Finally, if you point a character pointer to some characters, make sure they end with \0.

String Functions	Its Use
strlen	Returns number of characters in string.
strlwr	Converts all the characters in the string into lower case characters
strcat	Adds one string at the end of another string
strcpy	Copies a string into another
strcmp	Compares two strings and returns zero if both are equal.
strdup	Duplicates a string
strchr	Finds the first occurrence of given character in a string
strstr	Finds the first occurrence of given string in another string
strset	Sets all the characters of string to given character or symbol
strrev	Reverse a string

4.13 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] );
```

Check Your Progress 4

1. B

2. C

3. (a) I
   IGNOU

   (b) hello

   (c) hehe

Check Your Progress 5

1. D
2. C
3.
```
/*Program to concatenate two strings without using the strcat() function*/
#include<stdio.h>
#include<string.h>
main()
{
char str1[10];
char str2[10];
char output_str[20];
int i,j,k;
i=0;
j=0;
k=0;
printf("Input the first string: ");
gets(str1);
```

```
printf("\nInput the second string: ");
gets(str2);
while(str1[i]!='\0'
output_str[k++]=str1[i++];
while(str2[j]!='\0')
output_str[k++]=str2[j++];
output_str[k]='\0';
puts(output_str);
}
```

4. /* Program to find the string length without using the strlen() funtion */

```
#include<stdio.h>
#include<string.h>
main()
{
char string[60];
int len,i;
len=0;
i=0;
printf("Input the string : ");
gets(string);
while(string[i++]!='\0')
          len++;
printf("Length of Input String=%d",len);
getchar();
}
```

5. /* Program to convert the lower case letters to upper case in a given string without using strupp() function*/

```
#include<stdio.h>
main()
{
int i=0;
char source[10],destination[10];
printf("Input the string in lower-case");
gets(source);
while(source[i]!='\0')
{
  if((source[i]>=97)&&(source[i]<=122))
          destination[i]=source[i]-32;
      else
           destination[i]=source[i];
                 i++;
  }
  destination[i]=' \0 ';
  puts(destination);
}
```

4.14 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.

5. Computer Programming in C, *Raja Raman. V*, PHI, 2002.