UNIT 5: ARRAYS AND STRING HANDLING

UNIT STRUCTURE

5.0 Objectives
5.1 Introduction to arrays
5.2 Array Declaration
5.3 Accessing Elements of an Array
5.4 Array Initialization
5.5 One Dimensional Character Array
  5.5.1 String Declaration
  5.5.2 Different formatting techniques for displaying string
5.6 Passing one-dimensional array in function
  5.6.1 Passing single element of an array
  5.6.2 Passing an entire one-dimensional array to a function
5.7 Multi Dimensional Arrays
  5.7.1 Two Dimensional Arrays
  5.7.2 Initializing two-dimensional array
  5.7.3 Accessing two-dimensional array
  5.7.4 Memory Map of 2-dimensional array
  5.7.5 Passing Multi-dimensional arrays to function
5.8 Arrays of Strings
5.9 Built in String Functions
  5.9.1 strcat()
  5.9.2 strcpy()
  5.9.3 strlen()
  5.9.4 strcmp()
  5.9.5 strchr()
  5.9.6 Few more String Handling Library Functions
5.10 Summary
5.11 Questions for Exercises
5.12 Suggested Readings

5.0 OBJECTIVES

After completion of this unit you will be able to:

- Initialize and access array elements using subscript
- Write programs involving arrays and do searching and sorting
- Handle multi dimensional arrays
- Manipulation of strings using built-in string functions and various formatting techniques
- Write programs using concept of pointers and its manipulation
- Passing pointers to function
- Understand the concept of dynamic memory allocation

5.1 INTRODUCTION TO ARRAYS

An array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM. The elements of an array are of same data type and each item can be accessed using the same name.

If we deal with similar type of variables in more number at a time, we may have to write lengthy programs along with long list of variables. There should be more number of assignment statements in order to manipulate on variables. When the number of variables increases, the length of program also increase. In the above situations described above, where more

number of same types of variables is used, the concept of ARRAYS is employed in C language. These are very much helpful to store as well as retrieve the data of similar type. An Array describes a contiguously allocated non-empty set of objects with the same data type. The using arrays many number of same type of variables can be grouped together. All the elements stored in the array will referred by a common name.

For understanding the arrays properly, let us consider the following program:

```
main( )
{
int x ;
x = 5 ;
x = 10 ;
printf ( "\nx = %d", x ) ;
}
```

No doubt, this program will print the value of **x** as 10. Why so? Because when a value 10 is assigned to **x**, the earlier value of **x**, i.e. 5, is lost. Thus, ordinary variables (the ones which we have used so far) are capable of holding only one value at a time (as in the above example). However, there are situations in which we would want to store more than one value at a time in a single variable.C programming language provides a data structure called the array, which can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number1,numbers2, ..., and number100 to store 100 integer values, you can declare one array variable. An array declaration uses its size in [ ] brackets. For above example we can define an array as:

```
int   numbers[100];
```

Where numbers is defined as an array of size 100 to store 100 different values of integer data type. Each element of this collection is called an array-element and an integer value called the subscript is used to access specific element in an array. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element. As each integer value occupies 2 bytes, 200 bytes will be allocated for the whole array. If the memory address of the first element is 1501, then the memory addresses of the consecutive elements will be as follows:

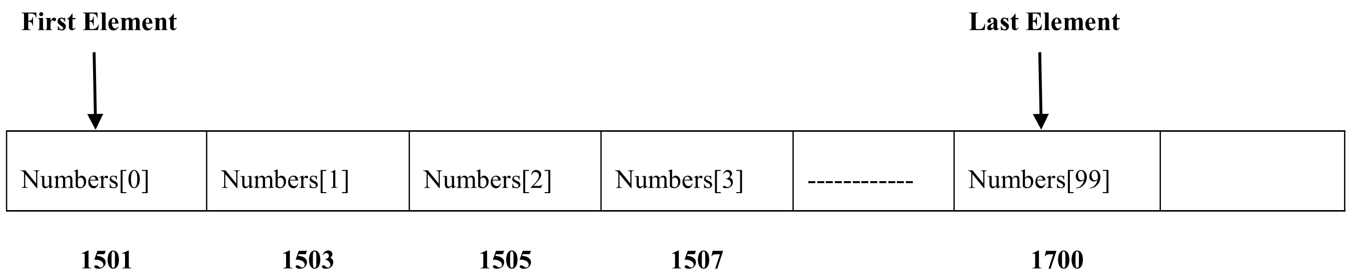

Fig 5.1:Representation of an array and it's memory allocation

In C, array elements are numbered from 0, so that an array of size N is indexed from 0 to $N - 1$. An array must contain at least one element, and it is an error to define an empty array.

```
double empty[0];  /* Invalid. Won't compile. */
```

5.3 Array Declaration

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows:

```
type arrayName [ arraySize ];
```

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement:

double balance[10];

Now *balance* is a variable array which is sufficient to hold upto 10 double numbers. The bracket ([ ]) tells the compiler that we are dealing with an array.

5.3 Accessing Elements of an Array

Once an array is declared, let us see how individual elements in the array can be referred. This is done with subscript, the number in the brackets following the array name. This number specifies the element's position in the array. All the array elements are numbered, starting with 0. Thus, **balance[2]** is not the second element of the array, but the third. In our program we are using the variable **i** as a subscript to refer to various elements of the array. This variable can take different values and hence can refer to the different elements in the array in turn. This ability to use variables as subscripts is what makes arrays so useful.

5.4 Array Initialization

So far we have used arrays that did not have any values in them to begin with. Let us now see how to initialize an array while declaring it. Following are a few examples that demonstrate this.

```
int num[6] = { 2, 4, 12, 5, 45, 5 } ;
int n[ ] = { 2, 4, 12, 5, 45, 5 } ;
float press[ ] = { 12.3, 34.2 -23.4, -11.3 } ;
```

Note the following points carefully:

(a) Till the array elements are not given any specific values, they are supposed to contain garbage values. This so happens because the storage class of these array is assumed to be **auto**. If the storage class is declared to be **static** then all the array elements would have a default initial value as zero.

(b) If the array is initialised where it is declared, mentioning the dimension of the array is optional as in the 2nd example above.

In C there is no check to see if the subscript used for an array exceeds the size of the array. Data entered with a subscript exceeding the array size will simply be placed in memory outside the array; probably on top of other data, or on the program itself.
This will lead to unpredictable results, to say the least, and there will be no error message to warn you that you are going beyond the array size. In some cases the computer may just hang. Thus, the following program may turn out to be suicidal.

```
main( )
{
int num[40], i ;
for ( i = 0 ; i <= 100 ; i++ )
num[i] = i ;
}
```

Thus, to see to it that we do not reach beyond the array size is entirely the programmer's botheration and not the compiler's.

Example 5.1 : A Simple Program Using Array
Let us try to write a program to find average marks obtained by a class of 30 students in a test.

```
main( )
{
    int avg, sum = 0 ;
    int i ;
    int marks[30] ; /* array declaration */
```

```
        for ( i = 0 ; i <= 29 ; i++ )
        {
              printf ( "\nEnter marks " ) ;
              scanf ( "%d", &marks[i] ) ; /* store data in array */
        }
        for ( i = 0 ; i <= 29 ; i++ )
               sum = sum + marks[i] ; /* read data from an array*/
        avg = sum / 30 ;
        printf ( "\nAverage marks = %d", avg ) ;
  }
```

5.5 ONE DIMENSIONAL CHARACTER ARRAYS

Character arrays are special. They have certain initialisation properties not shared with other array types because of their relationship with strings. Each character in the array occupies one byte of memory and the last character is always '\0'. Note that '\0' and '0' are not same. ASCII value of '\0' is 0, whereas ASCII value of '0' is 48.

The terminating null ('\0') is important, because it is the only way the functions that work with a string can know where the string ends. In fact, a string not terminated by a '\0' is not really a string, but merely a collection of characters.

5.5.1 String Declaration

```
char string[22] = {'N', 'a', 'l', 'a', 'n', 'd', 'a', 'O', 'p', 'e', 'n', 'U', 'n', 'i', 'v', 'e', 'r', 's', 'i', 't', 'y', '\0'};
                                    (or)

char string[22] = "NalandaOpenUniversity";
                                    (or)

char string []    = "NalandaOpenUniversity";
```

Difference between above declarations are, when we declare char as "string[22]", 22 bytes of memory space is allocated for holding the string value.

When we declare char as "string[]", memory space will be allocated as per the requirement during execution of the program.

5.5.2 Different Formatting Techniques for displaying strings

Example 5.5.2.1

```
/* Program to demonstrate printing of a string */
main( )
{
char name[ ] = "NalandaOpenUniversity" ;
int i = 0 ;
while ( i <= 22 )
{
printf ( "%c", name[i] ) ;
i++ ;
}
}
```

Output:
NalandaOpenUniversity

In the above example, we have initialized a character array, and then printed out the elements of this array within a **while** loop. We can also write the **while** loop without using the final value 24, because we know that each character array always ends with a '\0'.

Following program illustrates this.

```
main( )
{
    char name[ ] = " NalandaOpenUniversity " ;
    int i = 0 ;
    while ( name[i] != `\0' )
    {
        printf ( "%c", name[i] ) ;
        i++ ;
    }
}
```

This program doesn't rely on the length of the string (number of characters in it) to print out its contents and hence is definitely more general than the earlier one.

The same program can be written using the **printf( )** function as shown below.

```
main( )
{
        char name[ ] = " NalandaOpenUniversity " ;
printf ( "%s", name ) ;
}
```

The **%s** used in **printf( )** is a format specification for printing out a string. The same specification can be used to receive a string from the keyboard, as shown below.

```
main( )
{
char name[25] ;
printf ( "Enter your name " ) ;
scanf ( "%s", name ) ;
printf ( "Hello %s!", name ) ;
}
```

Note that the declaration **char name[25]** sets aside 25 bytes under the array **name[ ],** whereas the **scanf( )** function fills in the characters typed at keyboard into this array until the enter key is hit. Once enter is hit, **scanf( )** places a '\0' in the array. Naturally, we should pass the base address of the array to the **scanf( )** function. While entering the string using **scanf( )** we must be cautious about two things:

(a) The length of the string should not exceed the dimension of the character array. This is because the C compiler doesn't perform bounds checking on character arrays.

(b) **scanf( )** is not capable of receiving multi-word strings. Therefore names such as 'Shivansh Naman' would be unacceptable.

The way to get around this limitation is by using the function **gets( )**. The usage of functions **gets( )** and its counterpart **puts( )** is shown below.

```
main( )
{
```

```
char name[25] ;
printf ( "Enter your full name:" ) ;
gets ( name ) ;
puts ( "Hello!" ) ;
puts ( name ) ;
}
```

And here is the output...

```
Enter your name: Shivansh Naman
Hello!
Shivansh Naman
```

The program and the output are self-explanatory except for the fact that, **puts( )** can display only one string at a time (hence the use of two **puts( )** in the program above). Also, on displaying a string, unlike **printf( )**, **puts( )** places the cursor on the next line. Though **gets( )** is capable of receiving only one string at a time, the plus point with **gets( )** is that it can receive a multi-word string.

The ***printf*** function with %s format is used to display the strings on the screen. 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.

Example 5.5.2.2

Write a program to display the string "NALANDA" in the following format:

```
N
NA
NAL
NALA
NALAN
NALAND
NALANDA
NALAND
NALAN
NALA
NAL
NA
N
```

```
/* Program to display the sting in the above shown pattern*/

#include <stdio.h>
main()
{
        int x, y;
        static char string[]="NALANDA"
         printf("\n");
        for(x=0; x<7; x++)
        {
              y=x+1;
              printf("-7.*s \n", y, string);
/* reserves 7 character of space on to the monitor and minus sign is for left justified and for every loop the * is replaced by value of y which is incremented by 1 until it reaches to value 7 */
        }
```

```
    for (x=6; x>=0; x--)
    {
        y=x+1;
        printf("%-7.*s \n",y,string);
/* y value starts with 7 and for every loop it is decremented by 1 until it reaches to 1 */
    }
}
```

Output

```
N
NA
NAL
NALA
NALAN
NALAND
NALANDA
NALAND
NALAN
NALA
NAL
NA
N
```

5.6 **Passing One-dimensional Array In Function**

In C programming, a single array element or an entire array can be passed to a function.

5.6.1 Passing single element of an array

Single element of an array can be passed in similar manner as passing variable to a function.

Example 5.6.1

```
/* C program to pass a single element of an array to function */
#include <stdio.h>
void display(int age)
{
   printf("%d", age);
}

int main()
{
   int ageArray[] = { 2, 3, 4 };
   display(ageArray[2]); //Passing array element ageArray[2] only.
   return 0;
}
```

Output
4

5.6.2 Passing an entire one-dimensional array to a function

While passing arrays as arguments to the function, only the name of the array is passed (,i.e, starting address of memory area is passed as argument).

Example 5.6.2

```
/* C program to pass an array containing age of person to a function. This function should find average age and display
        the average age in main function. */

#include <stdio.h>
float average(float age[]);

int main()
{
   float avg, age[] = { 23.4, 55, 22.6, 3, 40.5, 18 };
   avg = average(age); /* Only name of array is passed as argument. */
   printf("Average age=%.2f", avg);
   return 0;
}

float average(float age[])
{
   int i;
   float avg, sum = 0.0;
   for (i = 0; i < 6; ++i) {
      sum += age[i];
   }
   avg = (sum / 6);
   return avg;
}
```

5.7 MULTI DIMENSIONAL ARRAYS

C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration –

```
type name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional integer array −

```
int threedim[5][10][4];
```

5.7.1 Two-dimensional Arrays

The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows −

```
type arrayName [ x ][ y ];
```

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns. A two-dimensional array **a**, which contains three rows and four columns can be shown as follows −

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

Figure 5.7.1

Thus, every element in the array **a** is identified by an element name of the form **a[i][j]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

5.7.2 Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example –

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

It is important to remember that while initializing a 2-D array it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.
Thus the declarations,
int arr[2][3] = { 12, 34, 23, 45, 56, 45 } ;
int arr[ ][3] = { 12, 34, 23, 45, 56, 45 } ;
are perfectly acceptable,
whereas,
int arr[2][ ] = { 12, 34, 23, 45, 56, 45 } ;
int arr[ ][ ] = { 12, 34, 23, 45, 56, 45 } ;
would never work.

5.7.3 Accessing Two-Dimensional Array Elements

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example −

```
int val = a[2][3];
```

The above statement will take the 4th element from the 3rd row of the array. You can verify it in the above figure 5.1.

Example 5.7.3

Here is a sample program that stores roll number and marks obtained by a student side by side in a matrix.

```
main( )
{
    int stud[4][2] ;
    int i, j ;
    for ( i = 0 ; i <= 3 ; i++ )
    {
          printf ( "\n Enter roll no. and marks" ) ;
```

```
        scanf ( "%d %d", &stud[i][0], &stud[i][1] ) ;
    }
    for ( i = 0 ; i <= 3 ; i++ )
            printf ( "\n%d %d", stud[i][0], stud[i][1] ) ;
}
```

5.7.4 Memory Map of a 2-Dimensional Array

Let us reiterate the arrangement of array elements in a two dimensional array of students, which contains roll nos. in one column and the marks in the other.

The array arrangement shown in Figure 5.1 is only conceptually true. This is because memory doesn't contain rows and columns. In memory whether it is a one-dimensional or a two-dimensional
array the array elements are stored in one continuous chain. The arrangement of array elements of a two-dimensional array in memory is shown below:

	Stud[0][0]	Stud[0][1]	Stud[1][0]	Stud[1][1]	Stud[2][0]	Stud[2][1]	Stud[3][0]	Stud[3][1]
Memory address	65508	65510	65512	65514	65516	65518	65520	65522

Figure 5.7.4 : Memory map of a two-dimensional array

We can easily refer to the marks obtained by the third student using the subscript notation as shown below:

```
printf ( "Marks of third student = %d", stud[2][1] ) ;
```

The C language embodies an unusual but powerful capability—it can treat parts of arrays as arrays. More specifically, each row of a two-dimensional array can be thought of as a one-dimensional array. This is a very important fact if we wish to access array elements of a two-dimensional array using pointers.
Thus, the declaration,

```
int stud [4][2] ;
```

can be thought of as setting up an array of 4 elements, each of which is a one-dimensional array containing 2 integers. We refer to an element of a one-dimensional array using a single subscript. Similarly, if we can imagine **stud** to be a one-dimensional array then we can refer to its zeroth element as **stud[0]**, the next element as **stud[1]** and so on. More specifically, **stud[0]** gives the address of the zeroth one-dimensional array, **stud[1]** gives the address of the first one-dimensional array and so on. This fact can be demonstrated by the following program.

Example 5.7.4:

```
/* Demo: 2-D array is an array of arrays */
main( )
{
int s[4][2] = {
{ 1234, 56 },
{ 1212, 33 },
{ 1434, 80 },
{ 1312, 78 }
} ;
int i ;
for ( i = 0 ; i <= 3 ; i++ )
printf ( "\nAddress of %d th 1-D array = %u", i, s[i] ) ;
}
```

And here is the output...

```
Address of 0 th 1-D array = 65508
Address of 1 th 1-D array = 65512
Address of 2 th 1-D array = 65516
```

Address of 3 th 1-D array = 65520
Let's figure out how the program works. The compiler knows that **s** is an array containing 4 one-dimensional arrays, each containing 2 integers. Each one-dimensional array occupies 4 bytes (two bytes for each integer). These one-dimensional arrays are placed linearly (zeroth 1-D array followed by first 1-D array, etc.). Hence each one-dimensional arrays starts 4 bytes further along than the last one.

5.7.5 Passing Multi-dimensional Arrays to Function

To pass two-dimensional array to a function as an argument, starting address of memory area reserved is passed as in one dimensional array

#Example 5.7.5: Pass two-dimensional arrays to a function

```
#include <stdio.h>
void displayNumbers(int num[2][2]);
int main()
{
   int num[2][2], i, j;
   printf("Enter 4 numbers:\n");
   for (i = 0; i < 2; ++i)
      for (j = 0; j < 2; ++j)
         scanf("%d", &num[i][j]);
   // passing multi-dimensional array to displayNumbers function
   displayNumbers(num);
   return 0;
}

void displayNumbers(int num[2][2])
{
   // Instead of the above line,
   // void displayNumbers(int num[][2]) is also valid
   int i, j;
   printf("Displaying:\n");
   for (i = 0; i < 2; ++i)
      for (j = 0; j < 2; ++j)
         printf("%d\n", num[i][j]);
}
```

Output

```
Enter 4 numbers:
2
3
4
5
Displaying:
2
3
4
5
```

5.8 ARRAYS OF STRINGS

Besides data base applications, another common application of two dimensional arrays is to store an array of strings. In this section we see how an array of strings can be declared and operations such as reading, printing and sorting can be performed on them.

A string is an array of characters; so, an array of strings is an array of arrays of characters. Of course, the maximum size is the same for all the strings stored in a two dimensional array. We can declare a two dimensional character array of MAX strings of size SIZE as follows:

char names[MAX][SIZE];
Since names is an array of character arrays, names[i] is the character array, i.e. it points to the character array or string, and may be used as a string of maximum size SIZE - 1. As usual with strings, a NULL character must terminate each character string in the array. We can think of an array of strings as a table of strings, where each row of the table is a string.

Names[0]	s	h	i	v	a	n	s	h	\0
Names[1]	n	a	m	a	n	\0			
Names[2]	s	r	i	j	a	n	\0		
Names[3]	r	a	u	n	a	k	\0		

Figure 5.8: An array of strings

5.9 BUILT IN STRING FUNCTIONS:

The header file <*string.h*> supports all the string functions in C language. The following is a list of the common string managing functions in C.

5.9.1 strcat() function

The strcat( ) function in C language concatenates two given strings. It concatenates source string at the end of destination string. Syntax for strcat( ) function is given below.

*char * strcat (char * destination, const char * source);*

For example:

strcat (str2, str1); – str1 is concatenated at the end of str2.
strcat (str1, str2); – str2 is concatenated at the end of str1.

As you know, each string in C is ended up with null character (‘\0’). In strcat( ) operation, null character of destination string is overwritten by source string’s first character and null character is added at the end of new destination string which is created after strcat( ) operation.

Example 5.9.1

Write a program to read two strings and append the second string to the first string without using library function strcat()

```
#include <stdio.h>
#include <string.h>

int main( )
{
  char source[ ] = "  String Functions" ;
  char target[ ]= " C tutorial" ;
  int i, j;
  printf ( "\nSource string = %s", source ) ;
  printf ( "\nTarget string = %s", target ) ;
```

```
  i=strlen(target);   // length of  target is stored in the variable i

   //  following loop extracts jth character from source and appends it at ith position of target

 for (j = 0; source[j] != '\0'; i++, j++) {
     target[i] = source[j];
  }

  s1[i] = '\0';

  printf ( "\nTarget string after strcat( ) = %s", target ) ;

}
```

OUTPUT:

```
Source string                  =  String Functions
Target string                  = C tutorial
Target string after strcat( ) = C tutorial String Functions
```

5.9.2 strcpy() function

strcpy( ) function copies contents of one string into another string. Syntax for strcpy function is given below.

*char * strcpy (char * destination, const char * source);*

For Example:

strcpy (str1, str2) – It copies contents of str2 into str1.
strcpy (str2, str1) – It copies contents of str1 into str2.

If destination string length is less than source string, entire source string value won't be copied into destination string.

For example, consider destination string length is 20 and source string length is 30. Then, only 20 characters from source string will be copied into destination string and remaining 10 characters won't be copied and will be truncated.

Example 5.9.2

Write a program to copy one string to another string without using library function strcpy()

```
#include<stdio.h>
#include<string.h>

int main() {
   char str1[100];
   char str2[100];
   int i;
   printf("\nEnter the String 1 : ");
   gets(str1);
```

```
  i=strlen(str1); // i stores the length of str1

   for (i = 0; str1[i] != '\0'; ++i)
  {
       str2[i] = str1[i];
  }
  str2[i] = '\0';

  //strcpy(str2, str1);

  printf("\nCopied String : %s", str2);

  return (0);
}
```

Output

Enter the String 1 : Patna

Copied String : Patna

5.9.3 strlen() function

strlen( ) function in C gives the length of the given string. Syntax for strlen( ) function is given below.

*size_t strlen (const char * str);*

strlen( ) function counts the number of characters in a given string and returns the integer value.
It stops counting the character when null character is found. Because, null character indicates the end of the string in C.

EXAMPLE 5.9.3

Write a program to count the length of the given string without using library function strlen()

```
#include <stdio.h>
#include <string.h>

int main( )
{
    int len;
    char array[20]="Nalanda" , ch;
    for (len = 0; array[len] != '\0'; len++)
    {
      ch = array[len];
      printf ( "character at  %d position  is : %c \n", len, ch);
    }

    // len = strlen(array) ;

    printf ( "\n \n string length  = %d \n" , len ) ;
    return 0;
}
```

OUTPUT:

character at 0 position is : N

character at 1 position is : a
character at 2 position is : l
character at 3 position is : a
character at 4 position is : n
character at 5 position is : d
character at 6 position is : a

string length = 7

5.9.4 strcmp() function

strcmp( ) function in C compares two given strings and returns zero if they are same.

If length of string1 < string2, it returns < 0 value. If length of string1 > string2, it returns > 0 value. Syntax for strcmp( ) function is given below.

*int strcmp (const char * str1, const char * str2);*

strcmp( ) function is case sensitive. i.e, "A" and "a" are treated as different characters.

EXAMPLE 5.9.4

Program to compare two strings

In this program, strings "fresh" and "refresh" are compared. 0 is returned when strings are equal. Negative value is returned when str1 < str2 and positive value is returned when str1 > str2.

```
#include <stdio.h>
#include <string.h>
int main( )
{
   char str1[ ] = "fresh" ;
   char str2[ ] = "refresh" ;
   int i, j, k ;
   i = strcmp ( str1, "fresh" ) ;
   j = strcmp ( str1, str2 ) ;
   k = strcmp ( str1, "f" ) ;
   printf ( "\n%d %d %d", i, j, k ) ;
   return 0;
}
```

OUTPUT:

0 -1 1

5.9.5 strchr() function

strchr( ) function returns pointer to the first occurrence of the character in a given string. Syntax for strchr( ) function is given below.

*char *strchr(const char *str, int character);*

For example, if:
string= "This is a string for testing"
strchr (string, 'i') = 3

Example 5.9.5

The following example shows the usage of strchr() function.

```
#include <stdio.h>
#include <string.h>

int main ()
{
   const char str[] = "http://www.ctutorial.com";
   const char ch = '.';
   char *ret;

   ret = strchr(str, ch);

   printf("String after |%c| is - |%s|\n", ch, ret);

   return(0);
}
```

Output:

String after |.| is - |.ctutorial.com|

In this program, strchr( ) function is used to locate first occurrence of the character '.' in the string "www.ctutorial.com". Character '.' is located at position 4 and pointer is returned at first occurrence of the character '.' .

5.9.6 Few more String handling library functions

strncat() function

strncat( ) function in C language concatenates (appends) portion of one string at the end of another string. Syntax for strncat( ) function is given below.

*char * strncat (char * destination, const char * source, size_t num);*

Example :
strncat (str2, str1, 3); – First 3 characters of str1 is concatenated at the end of str2.
strncat (str1, str2, 3); – First 3 characters of str2 is concatenated at the end of str1.

strlwr() function

strlwr( ) function converts a given string into lowercase. Syntax for strlwr( ) function is given below.

*char *strlwr(char *string);*

Example:
Let, str= “PATNA”
strlwr(str) = “patna”

strupr() function

strupr( ) function converts a given string into lowercase. Syntax for strlwr( ) function is given below.

*char *strupr(char *string);*

Example:
Let, str= “patna”
strupr(str) = “PATNA”

strrev() function

strrev( ) function reverses a given string in C language. Syntax for strrev( ) function is given below.

*char *strrev(char *string);*

Example:
Let, str= “patna”
strrev(str) = “antap”

5.10 Summary

An array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM.

Each element of this collection is called an array-element and an integer value called the subscript is used to access specific element in an array.

In C, array elements are numbered from 0, so that an array of size N is indexed from
0 to $N - 1$.

Each character in the array occupies one byte of memory and the last character is always '\0'.

While passing arrays as arguments to the function, only the name of the array is passed (,i.e, starting address of memory area is passed as argument).

It is important to remember that while initializing a 2-D array it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.

Each row of a two-dimensional array can be thought of as a one-dimensional array.

To pass two-dimensional array to a function as an argument, starting address of memory area reserved is passed as in one dimensional array.

A string is an array of characters; so, an array of strings is an array of arrays of characters.

There are various in built string handling function which can be used for string manipulation. These are stored in ***string.h*** header file.

5.11 Questions for Exercise

1. Fill in the blanks:
   a. "A" is a ___________ while 'A' is a ____________.
   b. A string is terminated by a ______ character, which is written as ______.
   c. The array **char name[10]** can consist of a maximum of ______ characters.
   d. The array elements are always stored in _________ memory locations.

2. Define an Array. Explain one-dimensional array
3. Explain about two - dimensional array?
4. Explain how to declare, initialize array of char type. 04. Write a C program to sort a list of numbers.
5. Write a program that interchanges the odd and even components of an array.
6. Write a program to find if a square matrix is symmetric.
7. Write a program that calculates the average of an array of 10 floating-point values.
8. Explain String handling functions in C. Write a C program to concatenate the given two strings and print new string.
9. Write a program that replaces two or more consecutive blanks in a string by a single blank.
   For example, if the input is

   ```
   Grim   return   to  the        planet  of       apes!!
   ```

   the output should be

   ```
   Grim return to the planet of apes!!
   ```
10. Write a program to sort a set of names stored in an array in alphabetical order.

5.12 Suggested Readings

- *Programming in C*, Third Edition, Stephen G. Kochan
- *Let Us C*, Fifth Edition, Yashavant P. Kanetkar
- *The C programming language*, Brain W Kernighan, Dennis M Ritchie, PHI.
- *Programming with C*, Second Edition, Byron Gottfried, Tata McGraw Hill, 2003.
- *C. The Complete Reference*, Fourth Edition, Herbert Schildt, Tata McGraw Hill, 2002.
- *The C Primer*, Leslie Hancock, Morris Krieger, Mc Gravy Hill, 2013.