Chapter 3: Strings

3.1 INTRODUCTION

- A string is a collection of characters under a single name.
- Group of characters defined between double quotation marks is a string constant e.g. "FYBCA sci has C Programming".
- Strings are used in building readable programs.
- Many operations are performed on character strings like reading, writing, correcting, copying, comparing and extracting portion of string.

3.2 CONCEPT

- A string is a collection of characters in sequence.
- This collection is treated as a single data item.

3.2.1 Declaration and Definition of String

- Using char data type, we can declare a character.
- In C programming there is no special data type to declare a string.
- To declare a string char data type has to be used with an array of characters.
- The syntax of declaration of a string variable is,

  char string_name[size];

  The size determines number of characters in string_name.
  string_name is a variable name needs to be a valid C variable name.

  Example: char name[20];
  char address [50]:

- The above declaration is stored in memory as:

0	1	2	3	4	5	6	7	8	9	-	-	-	19	Name

0	1	2	3	4	5	6	7	8	9	-	-	-	49	Address

- If we store value in string name as "FYBCA SCIENCE”
- Compiler assigns a string to a character array name. It automatically supplies a null character '\0' at the end of string.
- Therefore, the size should be equal to max number of character in string plus one for null Character '\0'

The above declaration stores the name in memory is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	-	-	19
F	Y	B	C	A		S	C	I	E	N	C	E	\O	\O	\O	\O	\O

3.2.2 Initialization String Variables

- Character arrays can be initialized like numeric arrays.
- We can initialize array after declaration or initialized at the time of declaration.

Example: address[12] = "MUMBAI PUNE";
char city[20] ('B', 'A', 'N', 'G', 'L', '0', 'R', 'E', '\0');

- ➢ In first example, Address is declared with size 12 elements long as initialized strings contains 11 characters and one element space is provided for null terminator.
- ➢ We have used null terminator in first example as it implicitly supplies null terminator '\0'. Just need to keep one space for null terminator.

- ➢ In second example, we have listed its elements using curly bracket. Each character has to be enclosed in single quotation. All elements to be separated by using (,) operator.
- ➢ The last character is null terminator, which must be supplied explicitly as shown above.

- ➢ C permits to initialize a character array without giving size i.e. number of elements.
- ➢ In this case size of array will be determined automatically based on elements initialized.
  Example: char wish[ ] = ('B', 'E', 'S', 'T', '   ', 'W', 'I', 'S', 'H', 'E', 'S', '\0');

- ➢ It's possible to declare string array size larger.
  Example: char str[10] = "HELLO";
- ➢ It gets array of size 10, places to value "HELLO" in it adds null character after HELLO and initializes all other elements to NULL.

0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	\O	\O	\O	\O	\O

- ➢ Following declaration is wrong:
  char str[3]="Good";
  this will generate compile time error.
  char str3[5];
  str3="GOOD";
  is not allowed as we cannot separate the initialization from declaration.

3.2.3 Format Specifier

- ➢ We use %s format specifier to read in string of characters.
- ➢ We use %s with input function scanf.
  Example: char name[10];
  scanf("%s", name);
- ➢ In case of character arrays the ampersand (&) is not required before variable name like other type of variables.
- ➢ Scanf function terminates its input on first white space it finds.
- ➢ White space includes blanks, tabs. carriage returns, form feeds and new lines.
  Example: Hello everyone
- ➢ It reads only Hello into name array since the blank space after the word Hello will terminates heading of string with a null character.

Program 3.1: Program for format specifiers.

```
#include<stdio.h>
main()
```

```
{
        char ch;
        char name[10];
        printf("Enter a character \n"); -
        scanf("%c", &ch);
        printf("Enter name \n");
        scanf("%s", &name);
        printf("%c %s", ch, name);
    }
```

3.2.4 String Literals/Constants & Variables

- String literal is a sequence of zero or more characters.
- String literals are enclosed in double quotes " ".
- A string contains characters that are similar to character literals ie. plain characters, escape sequence and universal characters.
- It's possible to break a long line into multiple lines using string literals and separating them using white spaces.

  Example:
  ```
  "hello, friends"
  "hello, \friends"
  "hello," "f " "friends"
  ```
- String literals is also known as string constant or constant string.
- String literals are stored in C as an array of characters and terminated by null byte, i.e. "O'.
- Character strings are often used in programs to build meaningful programs.

String variables:

- String variables are stored as array of characters terminated by a null byte '\0'.
- String variables can be initialized either with individual character or more commonly with string literals.
  ```
  char str[] = ('h','e','l','l','o', '\0');
  char str1[]= "hello";
  char str1[20] = "hello";
  ```

3.3 READING AND WRITING FROM AND TO CONSOLE

3.3.1 Reading String from Console

1) Using scanf() function:

- Like other data type variables, string variables read from terminal using scanf() function.
- %s format specifier is used to read in a string of characters.

  Example: char address [10];
  ```
  scanf("%s", address);
  ```
- For character arrays the ampersand (&) is not required before the variable name.

- scanf function automatically terminates the string that is read with a null character and therefore the character array should be large enough to hold the input string plus the null character.

Program 3.2: Program to read a series of words i.e. strings from terminal using scanf function.

```
#include<stdio.h>
main()
{
        char str1[30], str2[30], str3[30], str4[30];
        printf("Enter string values \n");
        scanf("%s %s", str1, str2);
        scanf("%s %s", str3, str4);
        printf("\n string1=%s \n string2= %s \n", str1, str2);
        printf("\n string3=%s \n string4=%s \n", str3, str4);
}
```

Reading string specifying field width

- We can specify the field width using form %ws in scanf statement for reading a specified number of characters from Input string.
  Example: scanf("%ws", name);

- While using width field remember the things.
- The width is greater than or equal to the string read in. In this case entire string will be stored in string variable.
- The width is less than the string width in this case excess characters than width will be truncated and left unread.
  Example: char name[10];
  scanf("%5s", name);

- Let consider input string is NIT, will be stored as:

0	1	2	3	4	5	6	7	8	9
N	I	T	\O	\O	\O	\O	\O	\O	\O

i.e. width field is greater than specified width.

- Let consider input string is SATYAM, will be stored as 8

0	1	2	3	4	5	6	7	8	9
S	A	T	Y	A	M	\O	\O	\O	\O

i.e. field width is less than specified with so sixth character is truncated.

2) Using getchar() function:

- getchar functior. is used to read a single character from terminal.
- Using getchar we can read character from terminal and place it into character array.
- This needs to be done repeatedly to read in, line of text and stored in an array.

- The reading of character is terminated when newline character '\n' is entered so null character is then inserted at the end of string.
- The syntax of getchar is,

```
char ch;
ch = getchar();
```

- It has no parameters.

3) Using gets() function:

- gets function is convenient method of reading a string containing white spaces from terminal
- gets function is available in <stdio.h> header file.

  Syntax: gets (string_variable);

  where string_variable is parameter for which gets function reads string from keyboard until newline character is encountered then appends null character to string.
- gets function never skip white space i.e. white space is allowed while reading string.
- This is not possible with scanf where white space terminates string.

  Example: char line[80];

```
gets(line);
printf("%s", line);
```

  Reads a line of text from keyboard and displays it on screen.

3.3.2 Writing string to Console

1) Using printf() function:

- printf function is used to display string to screen.
- %s format specification is used to display i.e. an array of characters that is terminated by null character.

  Example: printf("%s", name);
- Above statement is used to display contents of array name.
- Along with %s we can specify width and we can specify precision for the array to display.

  Example: % 10.4

  which says that display first four characters in the field width of 10 columns.
- We can add '-' minus sign in the specification.

  Example: %-10.4 s

  The string will be displayed / printed in left justification.

Program 3.3: Program to store string "FYBCA SCIENCE" in array name.

```
#include<stdio.h>
void main()
{
	char name [20] = " FYBCA SCIENCE ";
	printf("\n \n");
	printf("%16s \n", name);
	printf("%5s \n", name);
	printf("%16.5s \n", name);
	printf("%16.6s\N", name);
	printf("%16.0s\N", name);
```

```
        printf("%.3s\N", name);
        printf("%s \n", name);
}
```

- When field width is less than the length of string, the entire string is displayed.
- The integer value on the right side of decimal point specifies the number of characters to be printed.
- When number of characters to be printed is specified S zero nothing is displayed.
- The minus sign in specification caused the string to be printed left-justified.
- The specification. %ns prints first n characters of the string.

2) Using putchar() functions

- Like getchar function putchar function is used to display values of character variable.
- Its syntax is:

```
        char ch = '4';
        putchar(ch);
```

- putchar requires one parameter.
- Using putchar repeatedly it's possible to display string of characters stored in array.

Example: char name [20] = " FYBCA SCIENCE ";

```
        for (i = 0; i < 10; i++)
        putchar(name[i]);
```

Program 3.4: Program for getchar and putchar functions.

```
#include<stdio.h>
void main()
{
        int c
        printf( "Enter a value:");
        c = getchar();
        printf( "\nYou entered: ");
        putchar(c);
}
```

3) Using puts() function:

- Using puts we can display string value.
- puts declared in header file<stdio.h>.
- This function syntax is:

```
        puts(array_name);
```

where, array name variable containing string value.

- This displays the value of string variable array_name.

Example: puts (name)

Program 3.5: Program for gets and puts functions.

```
#include<stdio.h>
void main()
```

```
{
        char str[100];
        printf("Enter a string:");

        gets(str);
        printf("\nYou entered: ");
        puts(str);
}
```

Program 3.6: Program for scanf and printf functions.

```
#include<stdio.h>
void main()
{
        char str[100];
        int i;
        printf("Enter a value:");
        scanf("%s %d", str, &i);
        printf("\nYou entered: %s %d ", str, i);
}
```

3.4 IMPORTANCE OF TERMINATING NULL CHARACTER

- In C programming compiler assigns a character string to character array, it automatically supplies null character '\0' at the end of string. This terminating character plays important role as follows.
- Like other data types string is not a data type in C. It is considered as a data structure stored in array
- String is variable-length structure and is stored in fixed length array.
- The array size is not always the size of string array size.
- Sometimes is smaller or sometime array size is much larger than string stored in it. Therefore, last element of array need not represent end of string. It is necessary to determine the end of string data.
- NULL character serves as end-of-string marker i.e. terminating character.
  Example: char string[] = ('H', 'E', 'L', 'L', 'O', '\0');
- The above statement defines array string of 5 elements.
  char str[10]="GOOD";

0	1	2	3	4	5	6	7	8	9
G	O	O	D	\O	\O	\O	\O	\O	\O

In above case 10 characters size is created where value is placed in it, terminates with null character and initialized to NULL another element.

3.5 STRINGS AND POINTERS

- Strings are character arrays, so they can be declared and initialized as,
  char name[5]="SITA";

- Compiler automatically inserts null character '\0' at the end of string.
- C supports alternate method to create string using pointer variables of the type char.
  Example: char *name = "SITA";
- This statement creates a string for literal and stores its address in Pointer variable name.
- Pointer name points to first character of string "SITA"
- It is allowed to do runtime assignment for giving values to string pointer name,

  Example: char *name;
  name="SITA";
- Above statement name="SITA" is not string copy statement as name is a pointer not a string.
- We can display contents of name using output functions like printf/puts.
  Example: printf("%s", name);
  puts (name);

- Along with "name" pointer Indirection/deference operator * is not allowed to use even it is pointer to string, it is also name of string.
- It is possible to use a pointer to access individual characters in a string.

Program: 3.7: Program using pointers to calculate length of string.

```
#include<stdio.h>
void main()
{
        char *name;
        int length;
        name="PUNE";
        char *ptr=name;
        printf("%s \n", name);
        while(*ptr != '\0')
        {
                printf("%C is stored at address %u \n", *ptr, ptr);
                ptr++;
        }

}
```

3.6 ARRAY OF STRINGS

- An array of strings is a two-dimensional array.
- Array of string is required in application where list of names is required like student list, employee list. It is even called as "table of string".
- A list of names can be treated as table of strings.
- Two-dimensional character array can be used to store to entire list.

Example: an array of string for students list,
char student [3] [5];

This is used to store list of 3 names, each of length not more than 5 characters as shown below:

	0	1	2	3	4
Student[0]	Y	A	S	H	\O
Student[1]	H	A	N	S	\O
Student[2]	R	A	M	\O	

- As declared above array of strings i.e. two-dimensional array, above declaration will be stored in table form.
- To access name of ith student in the list we have to write student [i-1].
- So student[0] denote "YASH", student[1] denotes "HANS" and so on.
- Array of strings can be treated as column of strings.

Program 3.8: Programs to sort list of names in alphabetical order using array of strings.

```
#include<stdio.h>
#include<string.h>
#define ITEMS 5
#define MAX 20
void main()
{
        char string[ITEMS] [MAX], temp[MAX];
        int m=0, n=0;
        printf("Enter nares of %d items \n", ITEMS);
        while(m<ITEMS)
          scanf("%s", string[m++]);
        for(m=1; m<ITEMS; m++)
        {
                for(n=0; n<ITEMS-m; n++)
                {
                        if(strcmp(string[n], string[n+1]) > 0)
                        {
                                strcpy(temp, string[n]);
                                strcpy(string[n], string[n+1]);
                                strcpy(string[n+1], temp);
                        }

                }
        }
printf("\n Alphabetical list is \n");
        for(m=0; m<ITEMS; m++)
```

```
            printf("\r %s", string[m]);
}
```

3.6.1 Array of Character Pointers

- An array of pointers is a two-dimensional array. This concept is important use of pointer in handling table of strings.
- Consider an array of strings,
  char name[3] [25];
- The above statement declares name table containing three names each with maximum 25 characters. This requires 75 bites storage space in memory.
- Most of the times allocated memory bites are not completely utilized. So instead of making each row with a fixed no. of characters, we can make it a pointer to string of varying length.
  Example:
  char *name[3] = {"Yash Bhoskar","Shrayas Bhoskar", "Nikhil Bhoskar"};
- Above declaration declares name to be an array of 3 pointers to characters where each pointer is pointing to a particular name as follows:

name[0] → Yash Bhoskar
name[1] → Shrayas Bhoskar
name[2] → Nikhil Bhoskar

- This declaration allocates only 41 bites which is sufficient to hold all characters as shown.

Y	A	S	h		b	h	o	s	k	a	r	\o			
S	H	R	a	y	a	s		b	h	o	s	k	a	r	\0
N	I	K	h	i	l		b	h	o	s	k	a	r	\0	

To print all three names following statements are used:

```
for(i=0; i <=2; i++)
        printf("%s \n", name[i]);
```

3.7 USER DEFINED FUNCTIONS

- We will learn to create our own string based function to get the length of the string, to reverse the string, to copy string , to compare two strings with or without checking case, to extract a portion of string (extracting substring), splitting string and many more.

//Program 3.9. Count the total number of character available in a string and without using strlen() function

```
#include <stdio.h>
int stringLength(char *);
void main()
{
        char str[100]={0};
        int length;
```

```
	printf("Enter any string");
	scanf("%s", str);
	length=stringLength(str);
	printf("String length is: %d\n",length);
}

int stringLength(char* txt)
{
	int i=0, count=0;
	while(txt[i++]!='\0')
		count+=1;
	 return count;
}
```

//Program 3.10: Read a string and copy the string into another without using stringCopy() function.

```
#include <stdio.h>
void stringCpy(char* s1, char* s2);
void  main()
{
	char str1[100],str2[100];
	printf("Enter string 1: ");
	scanf("%[^\n]s", str1);//read string with spaces
	stringCpy(str1, str2);
	printf("String 1: %s \nString 2: %s\n", str1, str2);
}
void stringCpy (char* s1, char* s2)
{
	int i=0;
	while(s1[i]!='\0')
	{
		s2[i]=s1[i];
		i++;
	}
	s2[i]='\0'; /*string terminates by NULL*/
}
```

//Program 3.11: Read a string from the user, and convert the string into lowercase and uppercase without using the library function.

```
#include <stdio.h>
void stringLwr(char *s);
void stringUpr(char *s);
void main()
{
	char str[100];
	printf("Enter any string: ");
	scanf("%[^\n]s", str);//read string with spaces
	stringLwr(str);
	printf("String after stringLwr: %s\n",str);
	stringUpr(str);
	printf("String after stringUpr: %s\n",str);
}
void stringLwr(char *s)
{
	int i=0;
	while (s[i]!='\0')
	{
		if(s[i]>='A' && s[i]<='Z')
			s[i]=s[i]+32;
		++i;
	}
}
 void stringUpr(char *s)
{
	int i=0;
	while (s[i]!='\0')
	{
		if(s[i]>='a' && s[i]<='z')
			s[i]=s[i]-32;
		++i;
	}
}
```

//Program 3.12: Read a two string from the user and concatenate them without using the library function.

```
#include <stdio.h>
#include <string.h>
#define MAX_SIZE 100
void stringCat (char *s1, char *s2);
void main()
{
	char str1[MAX_SIZE], str2[MAX_SIZE];
	printf("Enter string 1: ");
	scanf("%[^\n]s", str1);//read string with spaces
	//getchar();//read enter after entering first string
	printf("Enter string 2: ");
	scanf(" %[^\n]s", str2);//read string with spaces
	//scanf(" %s", str2);//read string with spaces
	stringCat(str1, str2);
	printf("\nAfter concatenate strings are:\n");
	printf("String 1: %s \nString 2: %s", str1,str2);
	printf("\n");
}
void stringCat (char *s1, char *s2)
{
	int len, i;
	len=strlen(s1)+strlen(s2);
	if(len>MAX_SIZE)
	{
		printf("\nCan not Concatenate 1");
		return;
	}
	len=strlen(s1);
	for(i=0;i< strlen(s2); i++)
		s1[len+i]=s2[i];
	 s1[len+i]='\0'; /* terminates by NULL*/
}
```
3.3 PREDEFINED FUNCTIONS IN String.h

1) **strlen() function**

- This function is used to count number of characters in a string after counting function returns the count.

  Syntax: int_variable = strlen(string);

where integer variable is the one which receives the value of length of string returned by strlen function.

- The string is the argument for which function counts no. of characters in the string which may be a string constant.
- A counting ends at 1st null character.

  Example: int x;
  char name[] = "Deepali";
  x = strlen(name);
- X will be 7 as the name string contains total 7 characters, so strlen returns value 7 assigned to integer variable X.

2) **strcpy() function:**

- This function is similar to string assignment operator.
- This function assigns contents of string2 to string1 where string2 may be a character array variable or a string constant.

  Syntax: strcpy(string1, string2);

  Example: strcpy(pune city, "PUNE");

  will assign the string "PUNE" to string variable city.
- Similarly, the statement.

  strcpy(cityl, city2)

will assign the contents of the string variable city2 to the string variable city1. The size of the array city1 should be large enough to receive the contents of city2.

3) **strcat() function:**

- This function is used to concatenate or join strings together.

  Syntax: strcat(string1, string2)
  where string1 and string2 are character arrays.
- After successful execution of strcat string2 is appended to string1.
- While combining 1st null character is removed at the end of string one and placing string2 thereafter. In this case string2 remains unchanged.

  Example: strcat(part1, part2);

  while using strcat make sure that size of string1 should be declared large enough to accommodate string2.
- strcat function also appends a string constant to a string variable.

  strcat(part1, "GOOD");

  C permits nesting of strcat function.

  Example: strcat(strcat(stringl, string2), string3);
- In concatenates all three strings together and the result is stored in string1.

4) strcmp() function:

- This function compares 2 strings and returns value 0 if they are equal

  Syntax: strcmp(string1, string2)

  string1 and string2 may be string variables or string constant.

  Example: strcmp(namel, name2);
  strcmp(namel, "John");
  strcmp("Rom", "Ram");

Program 3.13: Program to show use of all above pre-defined string functions:

```
#include<string.h>
main()
{
	char s1[20], s2[20],s3[20];
	int x, 11, 12, 13;
	printf("\n\nEnter two string constants \n");
	scanf("%s %s", s1, s2);
	/* comparing s1 and s2 */
	X = strcmp(s1, s2);
	if(x!=0)
	{
		printf("\n \n strings are not equal \n");
		strcat(s1,s2);			/*joining si and s1 */
	}
	else
		printf("In in strings are equal \n");

	//copying s1 and s3
	strcpy(s3, s1);

	//finding length of string
```

```
        l1=strlen(s1);
        12=strlen(s2);

        /* Output */
        printf("\n s1=%s \t length %d characters \n", s1, l1);
        printf("s2=%s \t length = %d characters \n", s2, l2);
}
```

5) strnset():

- It sets the portion of characters in a string to given character.

  Syntax: char *strnset (char *string, char C, int n);

  where string is original string. C is given character, n is no. of characters to be replaced.

  Program 3.14: Program to show use of strnset()

```
#include<stdio.h>
#include<string.h>
void main()
{
  char str[20]="FYBCA Sci";
  printf("\nOriginal string is %s",str);
 // printf("\nTest string after strnset: %s",strnset(str,'#',7));
 printf("\nTest string after strnset: %s",memset(str,'#',5));
}
```

6) strtok():

- strtok function is used for splitting a string by some delimiter.
- For example, we have comma separated list of items from a file and we want individual items in an array.
- Function splits string according to given delimiter and returns next token.
- It needs in a loop to get all tokens

  Syntax: char *strtok(char str[], const char *delims);

Program 3.16: Program by using strok() function.

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[] = "Jan,Feb,Mar,Apr,May,June";
char *token = strtok(str,",");
//printf("%s \n", token);
while(token!= NULL)
{
    printf("%s \n", token);
  token=strtok (NULL,"_");
}
```

```
        }
```

7) strcmpi():

- strmpic() function is same as strcmp() function. But this function negotiates case.
- In strcmpi "A" and "a" are treated same characters, whereas strmp() function treats "A" & "a" as different character.
- It's a non-standard function which may not be available in standard library in C
- strmpi() and strcmp() compass two given strings and returns zero if they are same.
- If length of string1 < string2 it return<0 value.
- If length of string1 > string2 it returns > 0 value.

  Syntax: int strcmpi(const char *str1, const char *str2);

Program 3.15: Program to show use of strcmpi().

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

    }
    Output:
        0-1 1
```

8) strrev() Function:

- This function reserves all the characters of a string except the terminating null character.

  Syntax: char *strrev(char *str);

NOTE : strrev() function will not work in linux

//Program 3.17: Program using strrev() library function.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str[30];
    puts("\nEnter a string:");
    gets(str);
```

```
        strrev(str);
        puts("After reversal the string is:");
        puts(str);
}
```

//Program without using strrev()

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

void strrev(char *str)
 {
   int length = strlen(str);
   int i;
   char temp;

   for (i = 0; i < length / 2; i++) {
      temp = str[i];
      str[i] = str[length - i - 1];
      str[length - i - 1] = temp;
   }
}

int main()
{
   char str[] = "hello";
   printf("Original string: %s\n", str);
   strrev(str);
   printf("Reversed string: %s\n", str);
   return 0;
}
```

9) strlwr() Function:

- This function converts all the letters in a string to lowercase.
  Syntax: char* strlwr (char *str);

NOTE : strlwr() function will not work in linux

//Program 3.18: Program using strlwr() library function.

```
#include<stdio.h>
#include<string.h>
void main()
{
        char str[30];
        puts("\nEnter a string:");
        gets(str);
        strlwr(str);
        puts("Lowercase string is:");
        puts(str);
}
```

//Without using strlwr

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

void strlwr(char *str)
 {
   while (*str != '\0')
  {
      *str = tolower(*str);
      str++;
   }
}

int main()
 {
   char str[] = "Hello World";
   printf("Original string: %s\n", str);
   strlwr(str);
   printf("String in lowercase: %s\n", str);
   return 0;
}
```

10) strupr() Function:

- This function converts all the letters in a string to uppercase.

  Syntax: char * strupr (char *str);

NOTE : strupr() function will not work in linux

//Program 3.18: Program using strupr() library function.

```
#include<stdio.h>
#include<string.h>
void main()
{
        char str[30];
        puts("\nEnter a string:");
        gets(str);
        strupr(str);
        puts("Lowercase string is:");
        puts(str);

}
```

//Without using strupr

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

void strupr(char *str)
 {
   while (*str != '\0')
   {
       *str = toupper(*str);
       str++;
   }
}

int main()
 {
   char str[] = "Hello World";
   printf("Original string: %s\n", str);
   strupr(str);
   printf("String in uppercase: %s\n", str);
   return 0;
}
```

11) strchr() Function:

- This function scans a string for the first occurrence of a given character.

  Syntax: char * strchr (const char *str, int c);

```
//Program 3.20: Program using strchr() library function.
#include <stdio.h>
#include <string.h>

main()
{
   char str[] = "Hello, world!";
   char *ptr;
  ptr = strchr(str, 'o');
   if (ptr != NULL)
       printf("Found 'o' at position %ld\n", ptr - str);
   else
       printf("Character 'o' not found\n");
}
```

12) strrchr() Function:

- This function locates the last occurrence of a character in a given string.

  Syntax: char * strrchr (const char *str, int c);

Program 3.21: Program using strrchr() library function..

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

main()
{
   char str[] = "Hello, world!";
   char *ptr;
   ptr = strrchr(str, 'o');
   if (ptr != NULL)
      printf("Found 'o' at position %ld\n", ptr - str);
   else
      printf("Character 'o' not found\n");
}
```

13) strstr() Function:

- This function finds the first occurrence of a string in another string.
  Syntax: char * strstr (const char *str1, const char *str 2);

Program 3.22: Program using strstr() library function.

```
#include<stdio.h>
#include<string.h>
void main()
{
        char *ptr;
        char str1[30];
        char str2[30];
        puts("\nEnter a string:");
        gets(str1);
        puts("Enter the string to be found:");
        gets(str2);
        ptr=strstr(str1, str2);
        if(ptr==NULL)
                puts("String not found");
        else
                printf("Found %s in %s",ptr,str1);
}
```

14) strncpy() Function:

- This function copies at the most n characters of a source string to the destination string
  Syntax: char * strncpy (char * dest, const char * src, int n);

Program 3.23: Program using strncpy() library function.

```
#include<stdio.h>
#include<string.h>
void main()
{
        char src[60];
        char dest[60];
        int n;
        puts("\nEnter source string:");
        gets(src);
        puts("Enter the value of n:");
        scanf("%d",&n);
        puts("Source string is:");
        puts(src);
        strncpy(dest, src,n);
        dest[n]='\0';
        puts("Destination string is:");
        puts(dest);
}
```

15) strncat() Function:

- This function concatenates a portion of one string with another. It also appends at the most n characters of a source string to a destination string.
  Syntax: char * strncat (char * dest, const char * src, int n);

```
//Program 3.24: Program using strncat() library function.
#include<stdio.h>
#include<string.h>
void main()
{
char dest[60];
char src[60];
int n;
puts("\nEnter strings:");
gets(dest);
gets(src);
puts("Enter the value of n:");
scanf("%d",&n);
puts("The strings are:");
puts(dest);
puts(src);
strncat(dest, src,n);
puts("After concatenation:");
puts(dest);
}
```

16) strncmp() Function:

- This function compares a portion of two strings.
  Syntax: int strncmp(const char *str1, const char* str2, int n);

```
//Program 3.25: Program using strncmp() library function.
#include<stdio.h>
#include<string.h>
void main()
{
        char str1[30], str2[30];
        int res,n;
        puts("\nEnter string 1:");
        gets(str1);
        puts("Enter string 2:");
        gets(str2);
        puts("Enter the value of n:");
        scanf("%d",&n);
        res=strncmp(str1, str2,n);
        if(res==0)
        puts("Strings portions are equal");
        else
        puts("Strings portions are not equal");
}
```

17) strncmpi() Function:

- This function compares a portion of two strings without case sensitivity.
  Syntax int strncmpi (const char*str1, const char *str2, int n),

Note: In Linux, there isn't a direct alternative for strncmpi() like there is for strcmpi(). However, you can achieve a case-insensitive comparison with strncmp() by converting both strings to lowercase or uppercase before comparing them.

3.9 COMMAND LINE ARGUMENTS argc AND argv

- Command line argument is a parameter supplied to a program when program is invoked.
- With this two parameters/arguments are passed one in argc and second is argv.
- argc is an argument counter i.e. it counts number of arguments on command line.
- argv is an argument vector and represents an array of character pointers that points to command line arguments.
- The size of array will be equal to value of argc.
- Example: c:> program FILE1, FILE2, FILE3.
- There are four parameters supplied on command line. So argc = 4.
- argv is an array of four pointers to strings as below:

argv[0] =program

argv[1] =FILE1

argv[2]= FILE2
argv[3]= FILE3

- To access command line arguments, declare main function with argc and argv parameters.

Syntax: main(int argc, char *argv[]) { }

- Always first parameter in command line is program name so argv[0] is always program name. So argv[0] is always program name.

Program 3.27: Program for command line argument.

```
#include <stdio.h>
#include <stdlib.h>
void main(int argc, char *argv[]) // command line arguments
{
        if(argc!=5)
        {
        printf("Arguments passed through command line not equal to 5");
        exit(0);
        }
        printf("\n Program name: %s \n", argv[0]);
        printf("1st arg: %s \n", argv[1]);
        printf("2nd arg: %s \n", argv[2]);
        printf("3rd arg: %s \n", argv[3]);
        printf("4th arg: %s \n", argv[4]);
        printf("5th arg: %s \n", argv[5]);
}
```