MODULE-IV: STRINGS and POINTERS

STRINGS

- ✓ **"A string is a sequence of characters enclosed within double quotes".**

or

- ✓ **"String is an array of characters and terminated by NULL character which is denoted by '\0'.**

- ✓ A string is stored as a sequence of characters in an array terminated by '\0' (NULL character).
  Ex: Consider the String "SHREYAS S".
- ✓ This string is stored in the form of an array as shown below:

S	H	R	E	Y	A	S		S	\0
0	1	2	3	4	5	6	7	8	9

$\longrightarrow$ Null character

Declaring String variables

- ✓ A string is declared like an array of characters.

Syntax:

```
char string_name[size/length];
```

Where,
char: data type used to declare the strings or characters.
string_name: It specifies the name of the given string.
size: The size or maximum length (number of characters including '\0') of the string is specified in square brackets.

- ✓ **Length of the String:** The 'length' is the number of characters stored in the string up to but not including the null character.

Example

1. char name[21];
   - ✓ Size of the string is 21, means that it can store up to 20 characters plus one null character.

2. char str[10];
   - ✓ Size of the string is 10, means that it can store up to 10 characters plus one null character.

Initializing the Strings

- ✓ We can initialize an array of characters (Strings) in the declaration itself.

Examples:

1. char a[9]={'C', 'O', 'M', 'P', 'U', 'T', 'E', 'R', '\0'};

- ✓ The compiler allocates 9 memory locations ranging from 0 to 8 and these locations are initialized with the characters in the order specified.

a→	C	O	M	P	U	T	E	R	\0
	0	1	2	3	4	5	6	7	8

2. char b[10]={'R', 'A', 'M', 'A', '\0'};

- ✓ The compiler allocates 10 memory locations ranging from 0 to 9 and these locations are initialized with the characters in the order specified. The remaining locations are automatically initialized to null-characters as shown below:

b→	R	A	M	A	\0	\0	\0	\0	\0	\0
	0	1	2	3	4	5	6	7	8	9

3. char b[ ]={'C', 'O', 'M', 'P', 'U', 'T', 'E', 'R', '\0'};

- ✓ For this declaration, the compiler will set the array size to the total number of initial values. i.e. 9. The characters will be stored in these memory locations in the order specified as shown below:

b→	C	O	M	P	U	T	E	R	\0
	0	1	2	3	4	5	6	7	8

4. char b[ ]= "COMPUTER";

- ✓ Here, the string length is 8 bytes. But string size is 9 bytes. So, the compiler reserves 8+1 memory locations and these locations are initialized with the characters in the order specified. The string is terminated by '\0' by the compiler.

b→	C	O	M	P	U	T	E	R	\0
	0	1	2	3	4	5	6	7	8

String Input / Output Functions

Token-Oriented I/O functions

- ✓ The Input/Output operations performed by scanf() and printf() functions are called token-oriented Input/Output.

Reading a String using scanf( )

- ✓ It is possible to read a string using "scanf()".
- ✓ The conversion specification for reading a string using scanf() is "%s".
- ✓ The scanf() function stops reading characters when it finds the first white space character (spaces, tabs and new line characters).

Example: char name[20];
```
printf("Enter the name:");
scanf("%s", name);
```

Printing a String using printf( )

- ✓ The 'printf()' function prints the given string (all the characters but not the null character).
- ✓ The printf() conversion specification for a string variable is "%s".

Example: char name[20];

```
        printf("Enter the name:\n");
        scanf("%s", name);
        printf("The entered name is:%s", name);
```

Output:

```
Enter the name:
Rama
The entered name is:
Rama
```

Example C Program: Write a C program to read and display a string using scanf() and printf().

```
#include<stdio.h>
void main()
{
        char str[20];
        printf("Enter your name:\n");
        scanf("%s",str);
        printf("Entered name is:%s\n",str);
}
```

```
Output:
Enter your name:
Bapuji
Entered name is:
Bapuji
```

Line-Oriented I/O functions

- ✓ The Input/Output operation performed by **gets()** and **puts()** functions are called line-oriented Input/Output.
- ✓ The prototypes for these functions are in "string.h".
- ✓ This type of I/O functions processes entire line (string with white spaces). Hence these are called line-oriented I/O.

Reading a String using gets( )

- ✓ **gets() function is used to read a sequence of characters (string) with spaces in between.**
- ✓ The 'gets()' function allows us to read an 'entire line' of input including whitespace characters.

Syntax

gets(string);

Example:

```
char name[20];
printf("Enter the name:");
gets(name);
```

Printing a String using puts( )

- ✓ **'puts()' function is used for printing the given strings.**
- ✓ Whenever we want to display a sequence of characters stored in memory locations on the screen, then puts() function can be used.

Syntax

puts(string);

```
        printf("The entered name is:\n");
        puts(name);
```

String Manipulation Functions

✓ The standard library 'string.h' contains many functions for the string manipulation.

Si.No	String Functions	Description of each function
1.	strlen(str)	Returns length of the string str
2.	strcpy(dest,src)	Copies the source string src to destination string dest
3.	strncpy(dest,src,n)	Copies n characters of the source string src to destination string dest
4.	strcat(s1,s2)	Append string s2 to string s1
5.	strncat(s1,s2)	Append first n characters of string s2 to string s1
6.	strcmp(s1,s2)	Compare two strings s1 and s2
7.	strncmp(s1,s2,n)	Compare n characters of two strings s1 and s2
8.	strrev(string)	Reverse the given string
9.	strupr(string)	Used to convert the string to uppercase
10.	strlwr(string)	Used to convert the string to lowercase

1. strlen(str) and sizeof(): The length and size of a string

✓ The 'strlen()' function can be used to find the length of the string in bytes.

✓ This function calculates the length of the string up to but not including the 'null character'.

Syntax

length=strlen(str);

Where,

'str' is a string.

'length' is an integer representing the length of 'str' in bytes excluding the null character.

Example: Write a C program to demonstrate the usage of strlen().

```
#include<stdio.h>
#include<string.h>
void main()
{
        char str[10]= "hello";
        int length;
```

```
                    length=strlen(str);
                    printf("Length of the string is=%d\n",length);
            }
```

Output: Length of the string is=5

✓ **The 'sizeof' operator can be used to determine the size of a declared string.**

Syntax

```
sizeof(str);
```

Where,
'str' is a string.

Example: Write a C program to demonstrate the usage of sizeof().

```
        #include<stdio.h>
        #include<string.h>
        void main()
        {
                char str1[25]= "RAMA";
                printf("Size of string is=%d",sizeof(str));
        }
```

Output: Size of string is= 25

Example: Write a C program to find the length of string without using strlen() function.

```
        #include<stdio.h>
        #include<string.h>
        void main()
        {
                char str[];
                int length=0;
                printf("Enter the string\n");
                gets(str);
                while(str[length]!='\0')
                {
                        length ++;
                }
        printf("Length of the string is=%d\n", length);
        }
```

```
Output:
Enter the string
Good morning
Length of the string is=12
```

2. strcpy ( ): String Copy

Syntax

```
strcpy(dest,src);
```

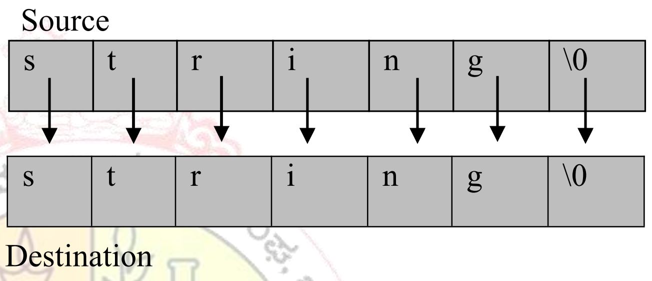
Where,
dest: it is the destination string
src: it is the source string.

- ✓ **The 'strcpy()' function copies the contents of source string src to destination string dest including '\0'.**
- ✓ The strcpy() function copies characters from the source string to the destination string until it finds null character.

Example: Write a C program to demonstrate the usage of strcpy().

```
#include<stdio.h>
#include<string.h>
void main()
{
        char src[10]= "string",dest[10];
        strcpy(dest,src);
        printf("The Source String=%s\n The Destination String=%s",src,dest);
}
```

Output: The Source String =string
The Destination String =string



Example: Write a C program to copy string without using strcpy() function.

```
#include<stdio.h>
#include<string.h>
void main()
{
        char src[100],dest[100];
        int i;
        printf("Enter the string\n");
        gets(str);
        for(i=0; str1[i]!='\0'; i++)
        {
                str2[i] = str1[i];
        }
        str2[i]='\0';
printf("copied string is=%d\n", str2);
 }
```

```
Output:
Enter the string
rahul
copied string is= rahul
```

3. strncpy(dest,src,n): String Number Copy

Syntax

strncpy(dest,src,n);

Where,

dest: it is the destination string.
src: it is the source string.
n: n is the number of characters to be copied into destination string.

- ✓ **'strncpy()' function is used to copy 'n' characters from the source string src to the destination string dest.**

Examples: Write a C program to demonstrate the usage of strncpy().

```
#include<stdio.h>
#include<string.h>
void main()
{
        char src[10]= "Computer", dest[10];
        strncpy(dest,src,3);
        printf ("The Source String=%s\n The Destination String=%s",src,dest);
}
```

Output:
```
The Source String=Computer
      The Destination String= Com
```

src

C	o	m	p	u	t	e	r	\0	
0	1	2	3	4	5	6	7	8	9

dest

C	o	m							
0	1	2	3	4	5	6	7	8	9

4. strcat(s1,s2): String Concatenate(Joining two strings together)

Syntax

strcat(s1,s2);

Where,

s1: It is the first string
s2: It is the second string

- ✓ **The 'strcat()' function is used to concatenate or join the two strings.**
- ✓ **The 'strcat()' function copies all the characters of string s2 to the end of string s1. The NULL character of string s1 is replaced by the first character of s2.**

Example: Write a C program to demonstrate the usage of strcat().

```
#include<stdio.h>
#include<string.h>
void main()
{
        char s1[15]= "Good";
```

```
                char s2[15]= “Morning”;
                strcat(s1,s2);
                printf(“The concatenated String=%s”,s1);
        }
```

Output:

```
                The concatenated String=GoodMorning.
```

s1

G	o	o	d	\0									
0	1	2	3	4	5	6	7	8	9	10	12	13	14

s2

M	o	r	n	i	n	g	\0							
)	1	2	3	4	5	6	7	8	9	10	11	12	13	14

s1

G	o	o	d	M	o	r	n	i	n	g	\0		
0	1	2	3	4	5	6	7	8	9	10	12	13	14

Example: Write a C program to demonstrate string concatenation without using strcat().

```
        #include<stdio.h>
        #include<string.h>
        void main()
        {
                char s1[15]= “Good”;
                char s2[15]= “Morning”, s3[20];
                int k,i;
                for (i=0; s1[i]!=’\0’; i++)
                {
                        s3[i] = s1[i];
                        k = k +1;
                }
                for (i=0; s2[i]!=’\0’; i++)
                {
                        s3[i] = s2[i];
                        k = k +1;
                }
                s3[k] = ‘\0’;
                printf(“The concatenated String=%s”,s3);
        }
```

5. strncat(s1,s2,n)- String Number Concatenate

Syntax **strncat(s1,s2,n);**

Where,
s1: It is the first string
s2: It is the second string
n: It is the number of characters of string s2 to be concatenated.

- ✓ **The 'strncat()' function is used to concatenate or join n characters of string s2 to the end of string s1.**

Example: Write a C program to demonstrate the usage of strncat().

```
#include<stdio.h>
#include<string.h>
void main()
{
        char s1[15]= "Good";
        char s2[15]= "Morning";
        strncat(str1,str2,4);
        printf("The concatenated String=%s",str1);
}
```

Output: The concatenated String=GoodMorn.

s1

G	o	o	d	\0									
0	1	2	3	4	5	6	7	8	9	10	12	13	14

s2

M	o	r	n	i	n	g	\0							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

s1

G	o	o	d	M	o	r	n	\0					
0	1	2	3	4	5	6	7	8	9	10	12	13	14

6. strcmp(s1,s2): String Compare

Syntax

```
strcmp(s1,s2);
```

Where,
s1: It is the first string.
s2: It is the second string.

- ✓ This function is used to compare two strings.
- ✓ The comparison starts with first character of each string. The comparison continues till the corresponding characters differ or until the end of the character is reached.

✓ The following values are returned after comparison:

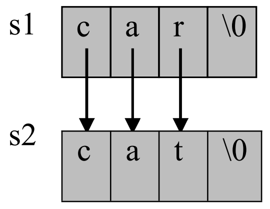
1. If two strings are equal, the function returns 0.

2. If s1 is greater than s2, a positive value is returned.

3. If s1 is less than s2, then the function returns a negative value.

Example:

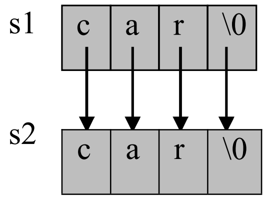
1.



strcmp(s1,s2);

✓ "car" and "cat" are different strings. The characters 'r' and 't' have different ASCII values. It returns **negative** value since ASCII value of 'r' is less than the ASCII value of 't'.

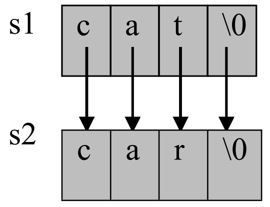
2.



strcmp(s1,s2);

✓ Since both the strings are same, the function returns 0.

3.



strcmp(s1,s2);

✓ "cat" and "car" are different strings. The characters 't' and 'r' have different ASCII values. It returns **positive** value since ASCII value of 't' is greater than the ASCII value of 'r'.

Example: Write a C program to demonstrate the usage of strcmp().

```
#include<stdio.h>
#include<string.h>
void main()
{
        char s1[10]="Hello";
        char s2[10]="Hey";
        if(strcmp(s1,s2)==0)
                printf("The two strings are identical");
         else
                printf("The two strings are not identical");
}
```

Example: Write a C program to compare two strings without using strcmp().

```
#include<stdio.h>
#include<string.h>
void main()
{         int i;
          char s1[10]="Hello";
          char s2[10]="Hey";
          len1 = strlen(s1);
          len2 = strlen(s2);
          if (strlen(s1) != strlen(s2))
                    printf("The two strings are different");
         else{
                    for (i=0; s1[i]!='\0'; i++)
                    {
                              if(s1[i]!=s2[i])
                    printf("strings are different");
                    break;
                    }
         }
                    printf("The two strings are identical");
}
```

7. strncmp(s1,s2,n): String Number Compare

Syntax **strncmp(s1,s2,n);**

Where,

s1: It is the first string.

s2: It is the second string.

n: It is the number of characters to be compared.

- ✓ This function is used to compare first n number of characters in two strings.
- ✓ The comparison starts with first character of each string. The comparison continues till the corresponding characters differ or until the end of the character is reached or specified numbers of characters have been tested.
- ✓ The following values are returned after comparison:

1. If two strings are equal, the function returns 0.

2. If s1 is greater than s2, a positive value is returned.

3. If s1 is less than s2, then the function returns a negative value.

Example: Write a C program to demonstrate the usage of strcmp().

```
#include<stdio.h>
#include<string.h>
void main()
{
        char s1[10]="Hello";
        char s2[10]="Hey";
        if(strcmp(s1,s2,2)==0)
                printf("The first two characters in the strings are identical");
        else
                printf("The first two characters in the strings are not identical");
}
```

POINTERS:

- A pointer is a derived data type in C. it is built from one of the fundamental data types available in C.
- A pointer is a variable that holds address of other variable. Since these memory addresses are the locations in the computer memory where program instructions and data are stored, pointers can be used to access and manipulate data stored in the memory.
- Although they appear little confusing and bit difficult to analyze for beginners, but they are a powerful tool and handy to use once they are mastered.

Declaration and Initialization of pointers

The operators used to represent pointers are:

- Address operator (&)
- Indirection operator (*)

Syntax:

```
ptr_data_type  *ptr_variable_name
ptr_variable_name = &variable_name  //where variable_name is the variable whose address has to
                                                          be stored in pointer.
```

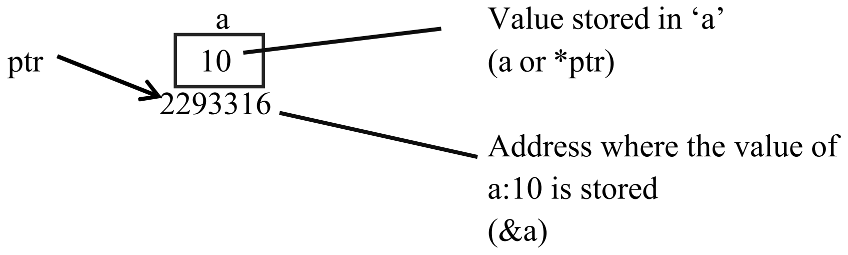
Example:

```
int a=10;
int *ptr;                //pointer declaration
then  ptr = &a           //
      *ptr = a;
```

i.e. ptr is a pointer holding address of variable ‘a’ and *ptr holds the value of a.

Example program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
      int a=10;
      int *ptr;
      ptr = &a;
      printf("%d\n",a);
      printf("%d\n",&a);
      printf("%d\n",ptr);
      printf("%d\n",*ptr);
      getch();
}
```



Pointers and Functions (Call by reference)

Call by reference method involves use of address of variables as actual parameters in calling function and pointer variable with (*) indirection operator is used at called function to perform required operations that is as formal parameters.

Consider an example of swapping two numbers using call by reference or using pointers.

Example:

```
#include<stdio.h>
#include<conio.h>
void swap(int *a, int *b);

void main()
{
        int x=10, y=20;
        printf("before swapping\n x=%d \n y=%d\n\n", x, y);
        swap(&x,&y);
        printf("after swapping: \n x = %d \n y = %d",x,y);
        getch();
}

void swap(int *a, int *b)
{
            int temp;
            temp=*a;
            *a=*b;
            *b=temp;
}
```

Pointers and arrays:

The operations performed using array can also be done using pointers.

Syntax:

```
data_type *ptr_name;
ptr_name = &array_name    or          ptr_name = array_name or
```

Here pointer does not point to all the elements of an array, instead initially it points to the first element of an array later which is incremented to get other elements.

```
Example:  int a[10]={11,12,13,14};
          int *ptr;
          ptr=&a
              or
          ptr=a; here ptr is initially pointing to 11It can be explained using below program,
```

```
/*program to illustrate pointer to array */
#include<stdio.h>
#include<conio.h>
void main()
{
int a[10]={11,12,13,14};
int *ptr,i;
ptr=a; //initially pointing to 1st element
for(i=0;i<4;i++)
{

        printf("%d\t",a[i]);
        printf("%d\n",&a[i]);
        printf("%d\t",*ptr);
        printf("%d\n",ptr);
        ptr++;  //makes ptr to point to next value by adding ptr= ptr+1
}
getch();
}
```