Basics -

- **String** is a **set of characters** that is treated as a single data item.
- C **does not** have a **string data type** to work with strings.
- In C, a **string constant** is a **sequence of characters** enclosed in **double quotes**. The characters may be letters, numbers, special characters and blank space. For example, "Y", "Good Morning", "2019", "4+3", etc.
- In C, a string is represented as an **array of characters**.
- The **end of the string** is marked with a **null character ('\0')**.

Declaration of a String –

- **char name[30];**

  Here **name** is a **character array.** It can be used to store any name of **maximum 29 characters** long, because **end of string** must be **terminated** by the **null character('\0')**.

- **char city[15];**

  Here **city** is a **character array**. It can store a city name of **maximum 14 characters long**.

String Initialization –

- C allows string initialization in either of the following three forms –
  1. **char msg[5] = "GOOD";**
  2. **char msg[5] = {'G', 'O', 'O', 'D', '\0'};**
  3. **char msg[ ] = {'G', 'O', 'O', 'D', '\0'};**
- Representation in memory will look like –

G	O	O	D	\0

In form 1, the null character is automatically added at the end of the string.
In each case, the elements are –
msg[0] = 'G' , msg[1] = 'O' , msg[2] = 'O' , msg[3] = 'D', msg[4] = '\0'

Note –

1. char colour[3] = "RED"; ⟹ Null character is NOT added automatically.

R	E	D

2. char colour[4] = "RED"; ⟹ Null character is added automatically.

R	E	D	\0

Reading string using scanf( )

```
char msg[10];
scanf("%s", msg);
```

⇨ If we type **GOOD BYE** as input then only string **GOOD** is read into the array **msg.** Here blank space after word **GOOD** will stop reading of the string.

Reading the text using getchar( )

```
int main()
{
        char msg[10], ch;
        int i = 0;
        while((ch=getchar( )) != '\n')
        {
                msg[i] = ch;
                i++;
        }
        msg[i] = '\0';
        ……..
}
```

⇨ If we type **GOOD BYE** as input then text **GOOD BYE** is read into the array **msg.** Note that we must add null character at the end of string explicitly.

Reading the text using gets()

```
int main()
{
        char msg[10];
        gets(msg);
        ……..
}
```

⇨ If we type **GOOD BYE** as input then text **GOOD BYE** is read into the array **msg.** Note that we need not to add null character at the end of string, as it will be added automatically.

Printing string using printf( )

```
char msg[10] = "GOOD BYE";
printf("%s\n", msg);
```

⇨ printf() function with format specifier %s is used to print string (array of characters) terminated by null character. Here entire contents of array **msg** i.e. **GOOD BYE** will be displayed on the screen and then cursor will be moved to the beginning of the next line due to **\n** after %s.

Reading the text using getchar( )

```
int main()
{
        char msg[10] = "GOOD BYE";
        int i;
        f or(i = 0; i < 8; i++)
                putchar(msg[i]);
        putchar('\n');
}
```

⇨ We can use putchar() function repeatedly to display string stored in an array using a loop. Here **GOOD BYE** will displayed on the screen. Then **putchar('\n');** statement will move the cursor to the beginning of the next line on the screen.

Reading the text using gets()

```
int main()
{
        char msg[10] = "GOOD BYE";
        puts(msg);
}
```

⇨ Here **puts(msg);** statement will print string stored in array **msg** and then moves the cursor to the beginning of the next line on the screen.

Note : outputs in above 3 cases will be same including the cursor position after printing the string.

The C library provides a set of string-handling functions for string manipulation and several character-handling functions for character manipulation in the string.

Some of these library functions are used in the following example.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main() {
  char a[50],b[50];
  int i, l;
  printf("\n Enter a string : ");
  gets(a);
  printf("\n%s", a);
  l=strlen(a);
  printf("\nLength of string is %d", l);
  strcat(a," how are you");
  strcpy(b, a);
  printf("\n%s", b);
  l=strcmp(a,b);
  printf("\nResult of compare = %d", l);
  printf("\n%d",strcmp("Ajoy", "ajoy"));
  printf("\n%d",strcmp("ajoy", "ajoy"));
  printf("\n%d",strcmp("ajoy", "Ajoy"));
  printf("\n%d",strcmpi("Ajoy", "ajoy"));
  printf("\n");
  for(i =0; a[i] != '\0' ; i++)
          putchar(toupper(a[i]));
  return(0);
}
```

Output of printf statements

```
Enter a string : Hello
Hello
Length of string is 5
Hello how are you
Result of compare = 0
-32
 0
32
0

HELLO HOW ARE YOU
```

We will now give a set of C library functions available for string-handling and character-handling in a string for your ready reference in tabular form.

Library Function	Return data type	Purpose	Header file
strlen(s)	int	Returns number of characters in a string	string.h
strcpy(s1, s2)	char *	Copies string s2 to string s1 including '\0'.	string.h
strcat(s1, s2)	char *	Adds string s2 after string s1	string.h
strcmp(s1,s2)	int	Compares two strings. Returns a negative value if s1 < s2, zero if s1 = = s2, a positive value if s1 > s2	string.h
strcmpi(s1,s2)	int	Compares two string without regard to case, i,e, ignore case.	string.h
toupper(c)	int	Converts letter to upper case	ctype.h / stdlib.h
tolower(c)	int	Converts letter to lower case	ctype.h / stdlib.h
gets(s)	char *	Enters string s from standard input device(keyboard)	stdio.h
puts(s)	char *	Sends string s to standard output device (VDU)	stdio.h
isalpha(c)	int	Determines the argument is alphabet or not. Returns non-zero if true, otherwise zero.	ctype.h
isdigit(c)	int	Determines the argument is digit or not. Returns non-zero if true, otherwise zero.	ctype.h
islower(c)	int	Determines the argument is lowercase or not. Returns non-zero if true, otherwise zero.	ctype.h
ispunct(c)	int	Determines the argument is a punctuation character or not.	ctype.h
strncpy(s1,s2,n)	char *	Copies at most n characters of the string s2 to string s1.	string.h
strncat(s1,s2,n)	char *	Concatenates at most n characters of string s2 to the end of string s1.	string.h
strncmp(s1,s2, n)	int	Compares at most n characters of string s1 to string s2. Returns a **negative value** if s1 < s2, **zero** if s1 = = s2, a positive value if s1 > s2	string.h

==========

Homework :

Given a string

char str[ ] = "123456789";

Write a C program that displays **n** lines of the following pattern.

1
232
34543
4567654
567898765

Note that value of **n** must be between 1 and 5 (both inclusive).

Hint: You have to include the given declaration

char str[] = "123456789";

in your code and must use string **str** to display the pattern.