TCS201: Programming and Problem Solving
UNIT-1

Strings:

String is an array of characters stored in a contiguous memory location and is terminated by a null character '\0'.

For ex:
"LION" is a string with 5 characters including the null character stored inside the computer memory.

The above string can be represented as:

0	1	2	3	4
'L'	'I'	'O'	'N'	'\0'

Declaration of a String:
Strings are declared in C using the **char** data type.
For ex:
char name[5]; //string variable name can store maximum of 5
//characters including the NULL character denoted //as **'\0'**
• The above declaration can be represented as:

name[0]	name[1]	name[2]	name[3]	name[4]
				\0

| ← Garbage values → |

Initialization of a String:
For ex:
char s[9]="LION";

The above declaration can be represented as:

name[0]	name[1]	name[2]	name[3]	name[4]
'L'	'I'	'O'	'N'	'\0'

Example-1: Program to demonstrate initialization of a string.

```
#include<stdio.h>
void main()
{
char name[5]={ “LION”};

printf("Character in the array at First position: %c \n", name[0]);
printf("Character in the array at Second position: %c \n", name[1]);
printf("Character in the array at Third position: %c \n", name[2]);
printf("Character in the array at Fourth position: %c ", name[3]);
}
```

Output:

Character in the array at First position: **L**
Character in the array at Second position: **I**
Character in the array at Third position: **O**
Character in the array at Fourth position: **N**

Example-2: Program to demonstrate initialization a character string.

```
#include<stdio.h>
void main()
{
char name[5]={‘C’,’A’,’T’};

printf("Character in the array at First position: %c \n", name[0]);
printf("Character in the array at Second position: %c \n", name[1]);
printf("Character in the array at Third position: %c \n", name[2]);

}
```

Output:

Character in the array at First position: **C**
Character in the array at Second position: **A**
Character in the array at Third position: **T**

The difference between Example-1 and Example-2 is that it in the first program the string declaration is terminated by a null character ie '\0' automatically by the compiler.
While in the second example the compiler **does not add a null character** to the declaration because it is treated as a character array as shown below.

Initialization of a String in Example-2:
For ex:
char s[]={'C','A','T'};

The above declaration can be represented as:

name[0]	name[1]	name[2]
'C'	**'A'**	**'T'**

Reading & Printing of Strings
• The strings can be accepted from the user using the following formatted functions:
scanf() : to accept the string from the user
printf() : to print a string to the screen

• Example: Program to illustrate the use of scanf() and printf().

```
#include<stdio.h>
void main()
{
char str[15];
printf("Type a string: \n");
scanf("%s", &str);
printf("You typed the string:%s", str);
}
```

Output:
Type a string: LION
You typed the string: LION

Problem with scanf in reading a string and its solution:

scanf can read a string from the input stream until the first occurrence of space. Hence, to enable scanf to read a string until a newline character ie '\n', the following modification can be done to the scanf :

```
#include<stdio.h>
void main()
{
char str[15];
printf("Type a string: \n");
scanf("%s", &str);
printf("You typed the string:%s", str);
}
```

Output:

Type a string: **LION THE KING OF JUNGLE**
You typed the string: **LION**

After modifying the scanf statement:

```
#include<stdio.h>
void main()
{
char str[15];
printf("Type a string: \n");
scanf("%^[\n]s", &str);
printf("You typed the string:%s", str);
}
```

Output:

Type a string: **LION THE KING OF JUNGLE**
You typed the string: **LION THE KING OF JUNGLE**

Unformatted input/output String functions: gets, puts

- gets()--- to read a string from the user until the user enters a newline Character ie '\n' (presses Enter key)

  puts()--- to display a string to the screen

• Example: Program to illustrate the use of gets() and puts().

```
#include<stdio.h>
void main()
{
char str[15];
printf("Type a string: \n");
gets(str);     //same as scanf("%^[\n]s", str);
printf("\nYou typed: ");
puts(str);     //same as printf("%s", str);
}
```

Output:

Type a string: **Programming in C**

You typed: **Programming in C**

STRING MANIPULATION FUNCTIONS

• Whenever strings needs to be manipulated in a program manually it adds the extra lines of program code and also makes it a very lengthy and difficult to understand.

• To avoid this C supports a large number of string handling functions. There are many functions defined in <string.h> header file.

Sl. No.	Function Name & its meaning
1	strlen(s1): → Returns the length of the strng s1
2	strcpy(s1,s2) → Copies the string s2 into s1
3	strncpy(s1,s2,n) → Copies first n characters of string s2 into s1
4	strcat(s1,s2) → Concatenates/Joins the string s2 to the end of s1
5	strncat(s1,s2,n) → Concatenates/Joins first n characters of string s2 to the end of s1
6	strcmp(s1,s2) → compares string s1 with s2; if s1 is equal to s2 the return a value zero; if s1<s2 it returns a value less than zero; otherwise if s1>s2 it returns a value greater than zero.
7	strncmp(s1,s2,n) → compares first n characters of string s1 with s2; if s1 is equal to s2 the return a value zero; if s1<s2 it returns a value less than zero; otherwise if s1>s2 the it returns a value greater than zero.
8	strcmpi(s1,s2) → compares string s1 with s2 by ignoring the case (uppercase or lowercase); if s1 is equal to s2 the return a value zero; if s1<s2 it returns a value less than zero; otherwise if s1>s2 the it returns

	a value greater than zero.
9	strchr(s1,ch)→ Returns a pointer to the first occurrence of character ch in s1
10	strstr(s1,s2)→ Returns a pointer to the first occurrence of the string s2 in s1
11	strrev(s1)→ Returns the reverse string after reversing the characters of the string s1

12	strtok(s1,delimiter): // splits str into tokens separated by the delimiters. It needs a loop to get all the tokens and it return NULL when there are no more tokens.
	char *strtok(char str[], const char *delims);

strlen(s1)

• The function calculates & returns the length of a string str passed to it as an argument.

Example: Program to illustrate the use of strlen().

```
#include<string.h>
#include<stdio.h>
void main()
{
char str[20];
int len;
printf("Type a string:");
gets(str);
len=strlen(str);
printf("\nLength of the string %s is %d ", str,len);
}
```

Output:

Type a string: **COLOR**

Length of the string COLOR is 5

strcpy(s1,s2)

• This function copies the content of string s2 into another string s2.

• Example: Program to illustrate the use of strcpy().

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

```
void main()
{
char s1[20],s2[20];
printf("Enter A string: ");
gets(s2);
strcpy(s1,s2); //Content of string s2 is copied into string s2
printf("Copied string:");
puts(s1);
}
```

Output:

Enter string: **PROGRAMMING IN C**

Copied string: **PROGRAMMING IN C**

strncpy(s1,s2,n)

- This function copies first n characters of s2 into another string s1.
- Example: Program to illustrate the use of strcpy().

```
#include<string.h>
#include<stdio.h>
void main()
{
char s1[20],s2[20];
int n;
printf("Enter a string: ");
gets(s2);
printf("\nHow many characters to be copied:");
scanf("%d",&n);
strcpy(s1,s2,n); //Content of string s2 is copied into string s1
printf("Copied string:");
puts(s1);
}
```

Output:

Enter a string: **PROGRAMMING IN C**

How many characters to be copied: **7**

Copied string: **PROGRAM**

strcat(s1,s2)

- Joins two strings by copying the string s2 to the end of s1.

Example: Program to illustrate the use of strcat().

```
#include <stdio.h>
#include <string.h>
void main()
{
char s1[10], s2[10];
printf("Enter the First String:");
gets(s1);
printf("\n Enter the Second String:");
gets(s2);
strcat(s1,s2); //concatenates string s1 and s2 stores the final string in s1
printf("\nConcatenated String: ");
puts(s1); // final concatenated string is stored in s1
}
```

Output:

Enter the First String: **Tom**
Enter the Second String: **Jerry**
Concatenated String: **TomJerry**

strncat(s1,s2,n)

Joins first **n** characters of **s2** joins with string s1 and stores it into s1.
Example: Program to illustrate the use of strcat().

```
#include <stdio.h>
#include <string.h>
void main()
{
char s1[10], s2[10];
int n;
printf("Enter the First String:");
gets(s1);
printf("\n Enter the Second String:");
gets(s2);
```

```
printf(“\nHow many characters needs to be appended:”);
scanf(“%d”,&n);

strncat(s1,s2,n); //concatenates n characters of string 2 to string s1 and
        //stores the final string in s1
printf("\nConcatenated String: ");
puts(s1); // final concatenated string is stored in s1
}
```

Output:
Enter the First String: **Tic**
Enter the Second String: **Tac**
How many characters needs to be appended: **2**
Concatenated String: **TicTa**

strcmp(s1,s2)

• This function compares two strings s1 with s2 and returns 0 if both the strings are equal i.e s1==s2.
returns a value < 0 if s1<s2
returns a value > 0 if s1>s2

• Example-1: Program using strcmp() function.

```
#include <string.h>
#include<stdio.h>
void main()
{
char s1[30],s2[30];
printf("Enter the first string: ");
gets(s1);
printf("Enter the second string: ");
gets(s2);
if(strcmp(s1,s2)==0)
        printf("Both strings are equal");
else
        printf("Strings are unequal");
}
```

Output:
Enter the first string: **Program**

Enter the second string: **program**
Both strings are unequal

• Example-2: Program using strcmp() function.

```
#include <string.h>
#include<stdio.h>
void main()
{
char s1[30],s2[30];
printf("Enter the first string: ");
gets(s1);
printf("Enter the second string: ");
gets(s2);
if(strcmp(s1,s2)==0)
        printf("Both strings are equal");
else
        if(strcmp(s1,s2)>0)
        printf("First string is greater than second string”);
else
        printf("First string is less than second string”);

}
```

Output:
Enter the first string: **program**
Enter the second string: **Program**
First string is greater than second string

Enter the first string: **RAINBOW**
Enter the second string: **rainbow**
First string is less than second string

strncmp(s1,s2,n)
• This function compares n characters of two strings s1 with s2 and
returns 0 if both the strings are equal i.e s1==s2 or
returns a value < 0 if s1<s2 or
returns a value > 0 if s1>s2

• Example: Program using strncmp() function.

```
#include <string.h>
#include<stdio.h>
void main()
{
char s1[30],s2[30];
int n;
printf("Enter the first string: ");
gets(s1);
printf("Enter the second string: ");
gets(s2);
printf("\nHow many characters to be compared:");
scanf("%d",&n);

if(strncmp(s1,s2,n)==0)
        printf("Both strings are equal");
else
        printf("Strings are unequal");
}
```

Output:
Enter the first string: **progRam**
Enter the second string: **program**
How many characters to be compared: **4**
Both strings are equal

strcmpi(s1,s2)

• This function compares two strings s1 with s2 by ignoring the cases (uppercase or lowercase) and
returns 0 if both the strings are equal i.e s1==s2.
returns a value < 0 if s1<s2
returns a value > 0 if s1>s2

• Example: Program using strcmpi() function.

```
#include <string.h>
#include<stdio.h>
void main()
{
char s1[30],s2[30];
```

```
printf("Enter the first string: ");
gets(s1);
printf("Enter the second string: ");
gets(s2);
if(strcmpi(s1,s2)==0)
        printf("Both strings are equal");
else
       printf("Strings are unequal");
}
```

Output:
Enter the first string: **program**
Enter the second string: **PROGRAM**
Both strings are equal

- **strtok():** function **strtok()** splits a string by some delimiter such as ‘ (single quote) **.** (dot operator) “ (double quote) # $ ‘ ‘ (space) etc.

  char *strtok(char * str, const char * delimiter);

  -Splits the string **str** into tokens and terminates them with a null character.
  delimiter-Characters at the beginning and end of **str** are skipped. On each subsequent call **delimiter** may change.

- **Example:** To split a string using **strtok**

```
#include <stdio.h>
#include <string.h>
int main()
{   char str[] = "Happy$New$Year";
   // Returns first token Happy
   char* token = strtok(str, "$");

   // Loops and prints tokens until one of the delimiters is present in str[].
    while (token != NULL)
    {       printf("%s\n", token);
            token = strtok(NULL, " ");
    }
   return 0;
}
```

Output:

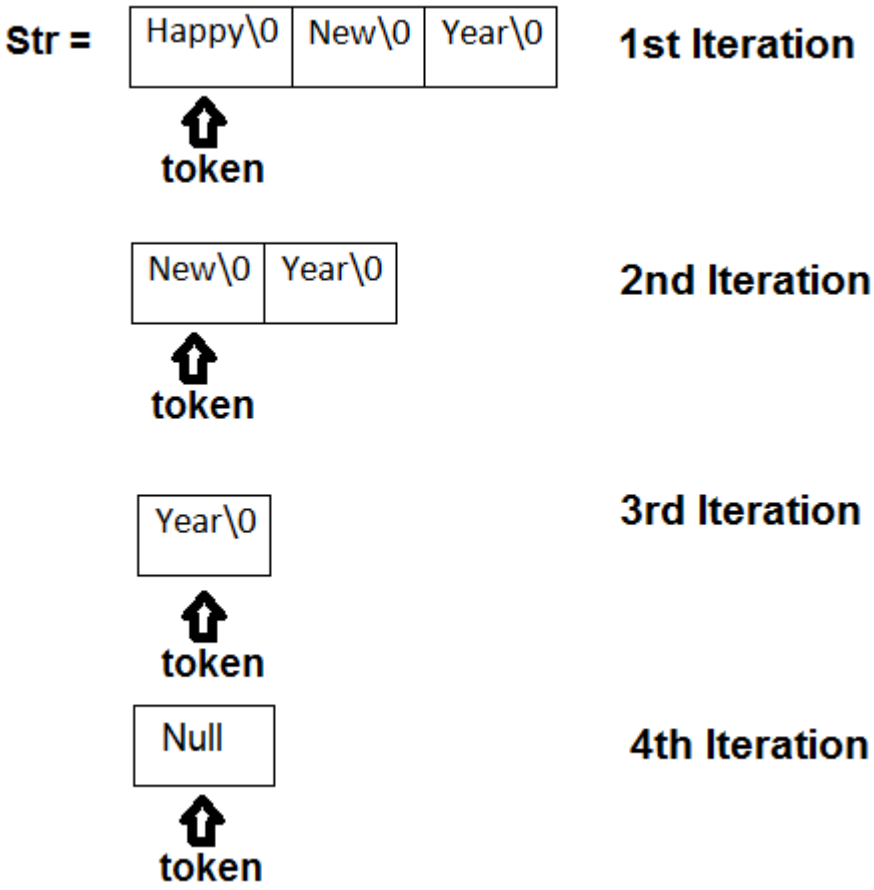
Happy
New
Year

Explanation:

strtok() stores the pointer in static variable where it last time left off & so on, in its 2nd call , when the NULL value is passed, strtok() gets the pointer from the static variable.

Please note **strtok()** is destructive i.e. it make changes to the original string. Hence, make sure you always have a copy of original one.

The below figure depicts the working of token on the **str** after each iteration and call to **strtok** in the ***while*** loop**.**



- **Programs on strings without using inbuilt functions**

1. Program to copy one string into another without using the inbuilt function.

```
#include<stdio.h>
void main()
{
        char s1[100], s2[100];
        int i;
        printf("Type a string:");
        gets(s1);
        i = 0;
        while (s1[i] != '\0')
        {
                s2[i] = s1[i];
                i++;
        }
        s2[i] = '\0';
        printf("Copied String is: %s ", s2);
}
```

Output:
Type a string : **Blueberry**
Copied String is: **Blueberry**

2. Program to compare two strings without using strcmp().

```
#include<stdio.h>
void main()
{
char s1[25],s2[25];
int i=0,flag=0;
printf("Enter the first string: ");
gets(s1);
printf("Enter the second string: ");
gets(s2);
while(s1[i]!='\0' && s2[i]!='\0')
{
        if(s1[i]!=s2[i])
        {       flag=1;
                break;
```

```
        }
         i++;
}
if (flag==0)
        printf("Both the strings are equal");
else
        printf("Both the strings are not equal");
}
```

Output:

Enter first string: Graphic Era

Enter second string: Graphic Era

Both the strings are equal

3. Example: Program to concatenate two strings without using inbuilt function.

```
void main()
{
char s1[25],s2[25];
int i=0,j=0;
printf(" Enter the First String:");
gets(str1);
printf("\n Enter the Second String:");
gets(str2);

while(s1[i]!='\0')
        i++;
s1[i++]=’ ‘;
while(s2[j]!='\0')
{
        s1[i]=s2[j];
        j++;
        i++;
}
s1[i]='\0';
printf("\n Concatenated String is %s",s1);
}
```

Output:

Enter the First String: **Merry**

Enter the Second String: **goround**

Concatenated String is **Merry goround**

Strings and Pointers:

Using character pointer strings can be stored in **read only string** in a shared segment.

When a string value is directly assigned to a pointer, in most of the compilers, it's stored in a **read-only block** (**generally in data segment**) that is shared among functions.

char *str="Hello World !!";

The above statement "**Hello World !!**" is stored in a shared **read-only segment**, but pointer **str** is stored in a **read-write memory**. **str** can be changed to point to something else but cannot change the value pointed to by str i.e. **doesnot allow to modify the string**. This kind of string should only be used when we don't want to modify string at a later stage in the program.

For Ex.

char str1[]="Make my day";

str=str1; **//allowed**

***str='W';** **//not allowed**

Results into Segmentation fault error message.

Ex. 1. Finding the length of a string using a pointer.

```
#include<stdio.h>
void main()
{
        char str[100];
        char *cptr;
        int len;
        printf("Type a string:");
        gets(str);
        cptr=&str[0];
```

```
        len = 0;

        while (*cptr != '\0')
        {
                len++;
        }
        printf("\nLength of the string is: %d ", len);
}
```

Output:

Type a string : **Above the Horizon**

Length of the string is: **17**

Ex. 2. Converting the Uppercase characters to Lowercase and vice-versa of a string using a pointer.

```
void main()
{
        char str[100];
        char *cptr;
        printf("Type a string:");
        gets(str);
        cptr=&str[0];

        printf("The converted string is:\n");
        while (*cptr != '\0')
        {

        if( isalpha(*cptr) )
        {
           if( isupper(*cptr) )
              printf("%c", tolower(*cptr));
                else
                   printf("%c", toupper(*cptr));
        }
        else
           printf("%c",*cptr);
```

```
        cptr++;
        }
}
```

Output:

Type a string : **An Apple a Aay Keeps the Doctor Away**

The converted string is:

aN aPPLE A aAY kEEPS THE dOCTOR aWAY

- **atoi function :** Converts a string argument to its integer type.

#include <stdlib.h>

int atoi(const char *cptr);

cptr : A pointer to a string to convert to an integer.

The atoi function returns the integer representation of a string.

Ex. 1. Program to convert a string into an integer using the inbuilt function atoi.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{  char a[3] = "277";
   int value = atoi(a);
   printf("Value = %d\n", value);
   return 0;
}
```

OUTPUT:

Value: 277