CE143: COMPUTER CONCEPTS & PROGRAMMING

UNIT-8
Character Arrays and Strings

N. A. Shaikh

nishatshaikh.it@charusat.ac.in

Topics to be covered

- **Introduction**
- **Difference of character array with numeric array**
- **Importance of NULL character**
- **Declaration and Initialization**
- **Various input and output methods of string**
- **formatted output of string**
- **Arithmetic operations on characters**
- **Various functions of string.h: strlen, strcat, strcmp, strcpy, strrev, strstr, etc**
- **Two dimensional character array (table of strings)**

Introduction

- A string is a **sequence of characters** that is treated as a single data item.

- Any group of characters defined between **double quotation mark** is a string constant.

Example:

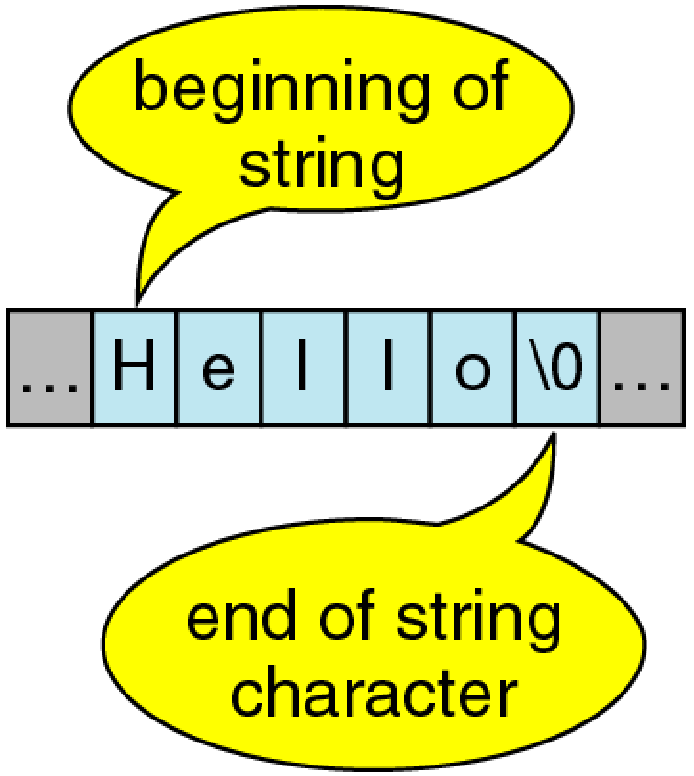
“Man is obviously made to think.”

Introduction

- **Strings** are one-dimensional array of characters terminated by a **null character '\0'**

For example:

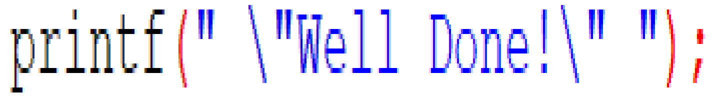
- The string "hello world" contains 12 characters including '\0' character which is automatically added by the compiler at the end of the string.



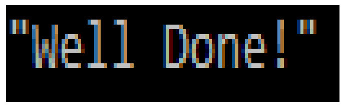
Introduction

- If we want to print **double quote** in the string, we may use **back slash** with it

- “ **\”** Man is obviously made to think **\”** said Pascal.”



```
printf(" \"Well Done!\" ");
```



```
"Well Done!"
```

Introduction

- The **common operations** performed on character strings :
  - Reading and writing strings
  - Combining strings together
  - Copying one string to another
  - Comparing strings for equality
  - Extracting a portion of a string

Declaration of String Variables

Syntax:

```
char string_name[size];
```

Example:

```
char city[10];
char name[30];
```

NOTE: When the compiler assigns a character string to a character, it **automatically** supplies a **null character ('\0')** at the end of the string. Therefore, the size should be equal to the maximum number of characters in the **string plus one**.

Initialization of String Variables

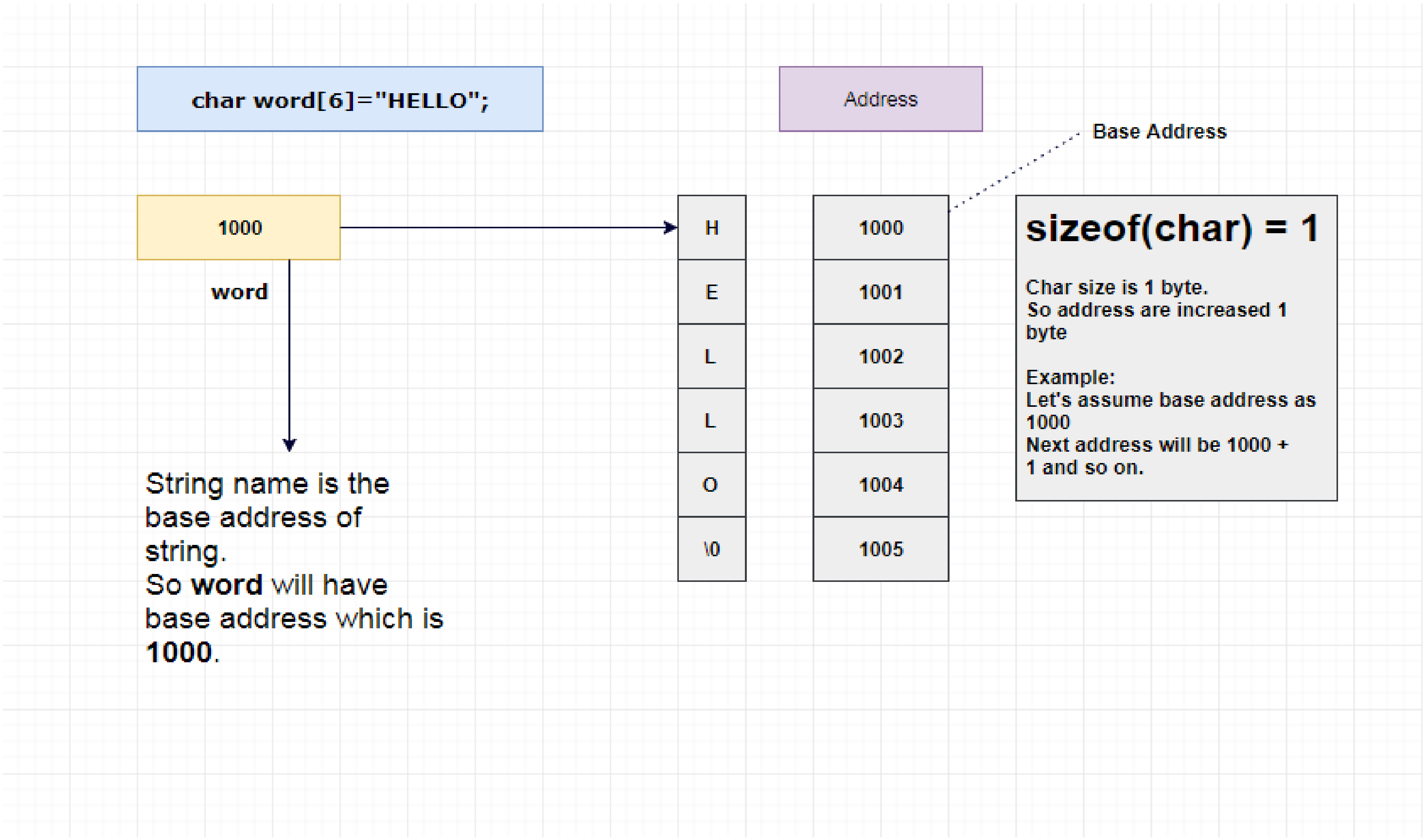
- C permits a character array to be initialized in either of the following two forms:

```
char city[9] = "NEW YORK";

char name[9]; = {'N','E','W',' ','Y','O','R','K','\0'};
```

NOTE:

1. String NEW YORK contains 8 characters and one element space is provided for the null terminator.
2. When character array is initialized by listing its elements, we must supply **null terminator explicitly**.

Initialization of String Variables



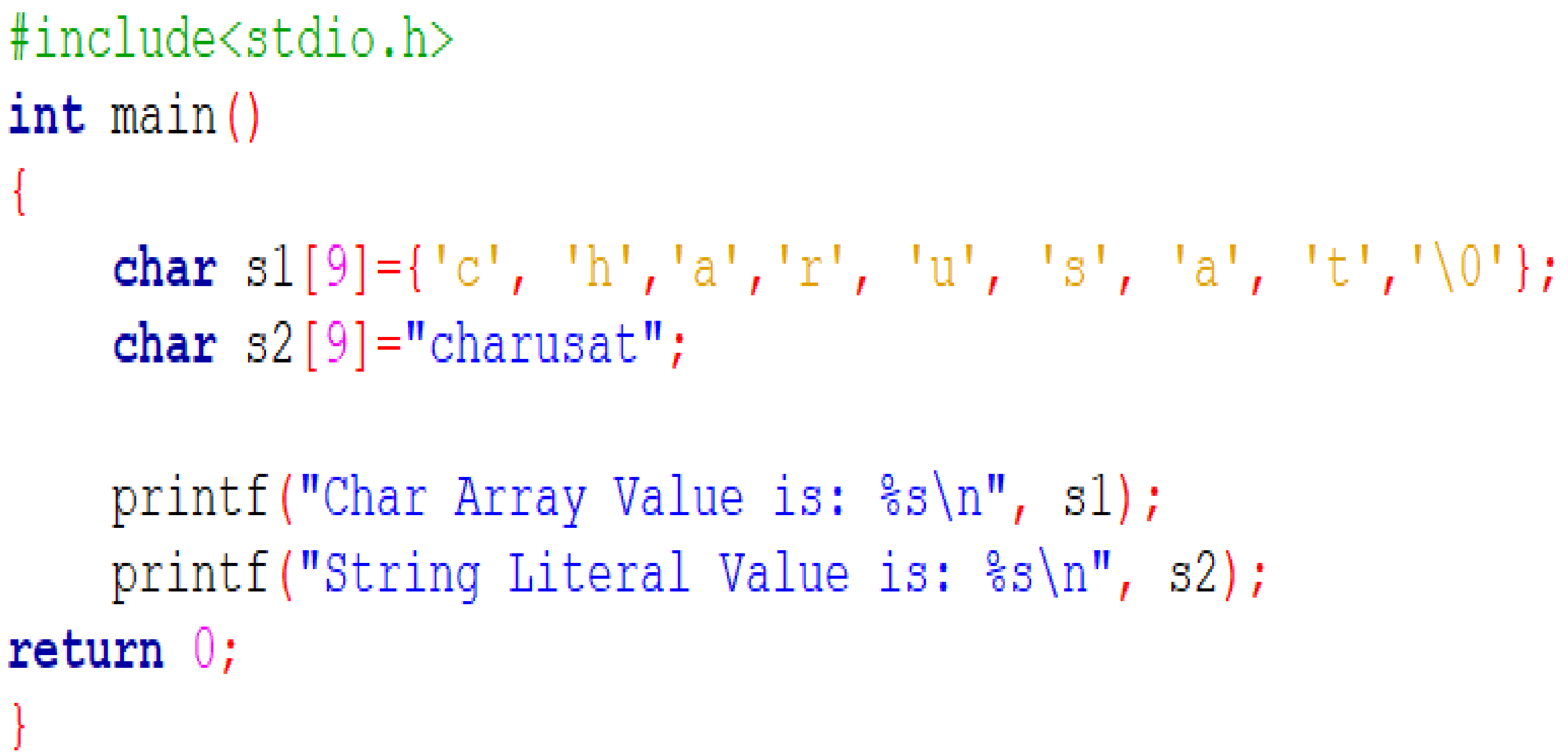
sizeof(char) = 1

Char size is 1 byte.
So address are increased 1 byte

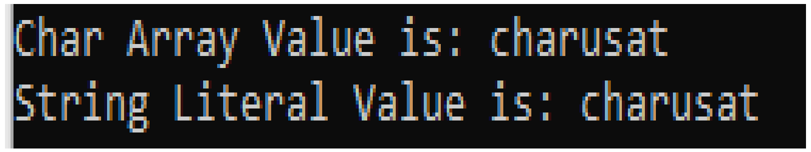
Example:
Let's assume base address as 1000
Next address will be 1000 + 1 and so on.

String name is the base address of string.
So **word** will have base address which is **1000**.

Initialization of String Variables



```c
#include<stdio.h>
int main()
{
    char s1[9]={'c', 'h','a','r', 'u', 's', 'a', 't','\0'};
    char s2[9]="charusat";

    printf("Char Array Value is: %s\n", s1);
    printf("String Literal Value is: %s\n", s2);
return 0;
}
```



```
Char Array Value is: charusat
String Literal Value is: charusat
```

Initialization of String Variables

C also permits to initialize a character array **without specifying the number of elements**.

```
char city[] = "NEW YORK";

char name[]; = {'N','E','W',' ','Y','O','R','K','\0'};
```

NOTE:

The size of the array will be determined automatically, based on the number of elements initialized.

Initialization of String Variables

We can also declare the **size much larger** than the string size in the initializer.

```
char string[10]="GOOD";
```

G	O	O	D	\0	\0	\0	\0	\0	\0

However, the following declaration is **illegal**:

```
char string[3]="GOOD";
```

warning: initializer-string for array of chars is too long

Initialization of String Variables

We cannot separate the initialization from declaration

```
char string[5];
string="GOOD";
```

error: assignment to expression with array type

Similarly, following is not allowed

```
char s1[4]="abc";
char s2[4];
s2=s1;   /*Error*/
```

error: assignment to expression with array type

NOTE: An array name cannot be used as the left operand of an assignment operator.

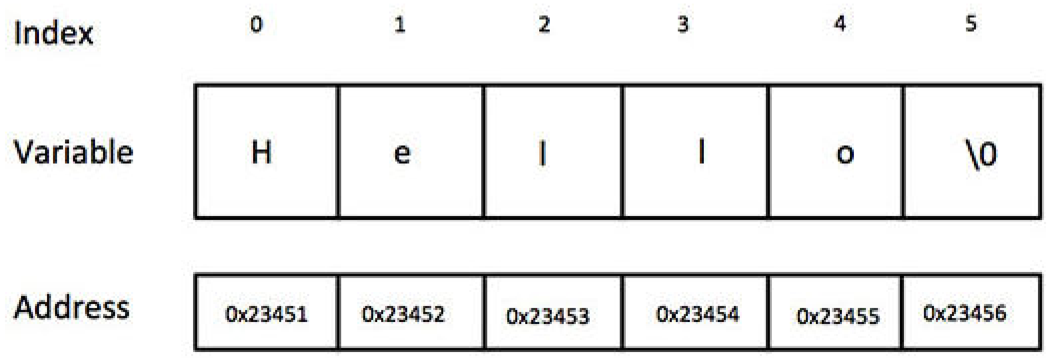
Need of NULL character

- The string is a **variable-length** structure and is stored in a **fixed-length** array.
- The array size is not always the size of the string and most often it is much larger than the string stored in it.
- Therefore, the last element of the array need not represent the end of the string.
- We need some way **to determine the end of the string** data and the null character serves as the **“end-of-string”** marker.

Importance of NULL character

- 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.
- Memory Representation of String:

```
char s[] = "Hello";
```



character array	numeric array
An "**character**" array is an array whose elements are all of an **char type**	An "**integer**" array is an array whose elements are all of an **int type**
char array takes **less memory** to allocate as compare to numeric array	int array take **more memory** to allocate
Syntax: char array[10];	**Syntax:** int array[10];
strings are terminated with sentinel char.	integer arrays are not terminated with any sentinel char.
size: number of elements*1 byte	**size:** number of elements* (2 byte or 4 byte) Depend on compiler

NOTE:

Also, a **char array** is **not** terminated with a sentinel character. **Strings** are terminated with a sentinel character. Strings (including the sentinel character) are saved in char arrays.
All strings are char arrays; not all char arrays are strings.

Reading Strings from Terminal

1. Using **scanf** function
2. Using **getchar** function(Used to read single character)
3. Using **gets** function

Reading Strings from Terminal: Using scanf

The familiar input function scanf can be used with,

1) %c format specification (Read character by character)

```
char str[10];
int i;

for(i=0;i<10;i++)
    scanf("%c",&str[i]);
```

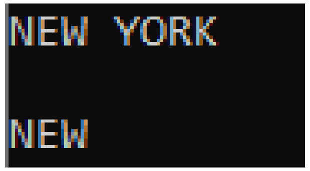
2) %s format specification (Read entire string)

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

NOTE: The ampersand(&) is not required before the variable name.

Reading Strings from Terminal: Using scanf

Problem: scanf function terminates its input on the first **white space**(blanks, tabs, carriage returns, form feeds and new lines) it finds.

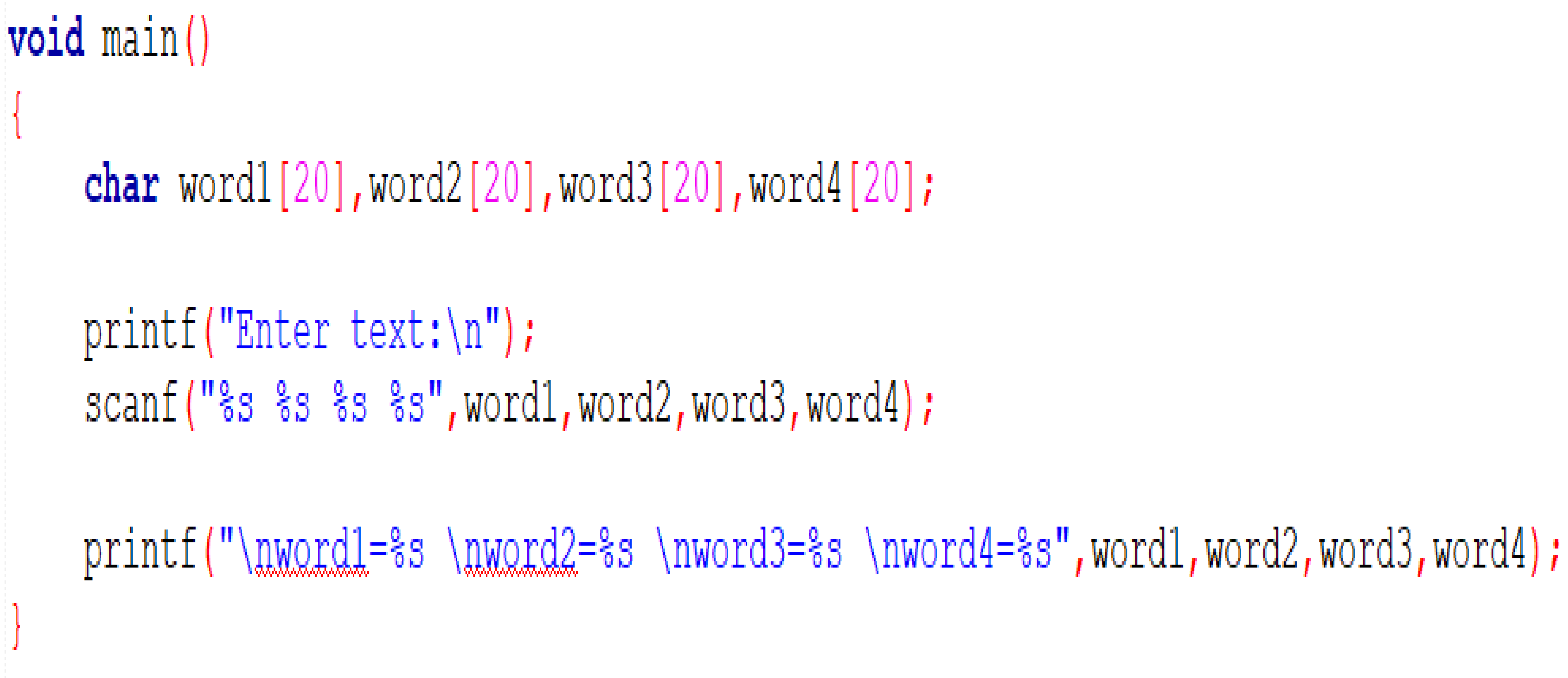

```
NEW YORK

NEW
```

If we want to read the entire “NEW YORK”, Then we may use two character arrays

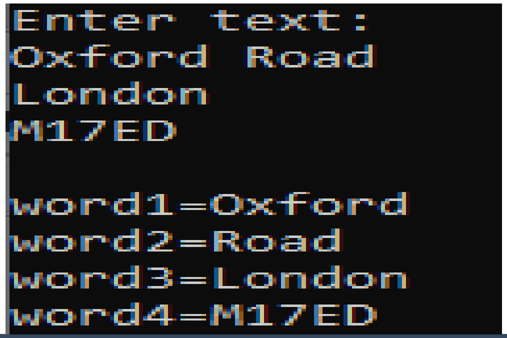
```
char str1[5],str2[5];
scanf("%s %s",str1,str2);
```

Will assign the string “NEW” to str1 and “YORK” to str2.

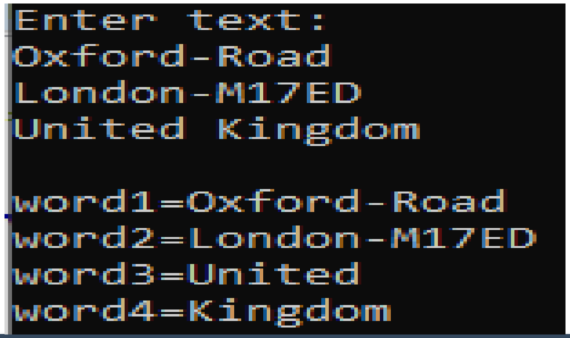
Reading Strings from Terminal: Using scanf



```
void main()
{
    char word1[20],word2[20],word3[20],word4[20];

    printf("Enter text:\n");
    scanf("%s %s %s %s",word1,word2,word3,word4);

    printf("\nword1=%s \nword2=%s \nword3=%s \nword4=%s",word1,word2,word3,word4);
}
```



```
Enter text:
Oxford Road
London
M17ED

word1=Oxford
word2=Road
word3=London
word4=M17ED
```



```
Enter text:
Oxford-Road
London-M17ED
United Kingdom

word1=Oxford-Road
word2=London-M17ED
word3=United
word4=Kingdom
```

Reading Strings from Terminal: Using scanf

We can also specify the **field width using %ws**

```
scanf("%ws",name);
```

Here, the two following things may happen:

1. The width w is equal to or greater than the number of characters typed in.

```
char name[10];
scanf("%5s",name);
```

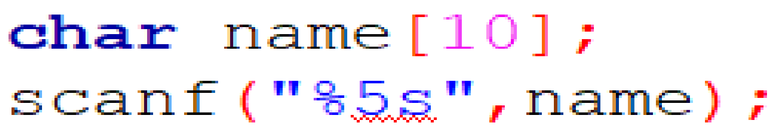
The input string RAM will be stored as:

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

The entire string will be stored in the string variable

Reading Strings from Terminal: Using scanf

2. The width w is less than the number of characters in the string.



```
char name[10];
scanf("%5s",name);
```

The input string KRISHNA will be stored as:

K	R	I	S	H	\0	?	?	?	?
0	1	2	3	4	5	6	7	8	9

The excess characters will be truncated and left unread.

Reading Strings from Terminal: Using scanf

Reading a Line of Text

- Using format specification known as the **edit set conversion code %[..]** that can be used to read a line containing a variety of characters, including whitespaces.

```
char line[80];
scanf("%[^\n]",line);
printf("%s",line);
```

Reading Strings from Terminal: Using getchar()

- **getchar()** as the name states **reads only one character** at a time.
- In order **to read a string**, we have to use this function **repeatedly** until a terminating character('\n') is encountered.
- The characters scanned one after the other have to be stored simultaneously into the character array.

Note: You have to **manually insert the null '\0' character** at the end of string using this method.

Reading Strings from Terminal: Using getchar()

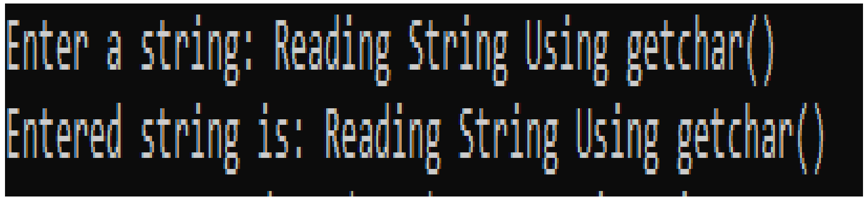
Syntax:

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

NOTE: getchar function has no parameters.

Reading Strings from Terminal: Using getchar()

```
void main()
{
    char str[50], ch;
    int i=0;
    printf("Enter a string: ");
    ch = getchar ();

    while(ch!='\n')
    {
        str[i] = ch;
        i++;
        ch = getchar();
    }
    str[i] ='\0';
    printf("Entered string is: %s", str);
}
```



```
Enter a string: Reading String Using getchar()
Entered string is: Reading String Using getchar()
```

Reading Strings from Terminal: Using gets()

Another method of reading a String of text containing whitespace is to use **gets** function from **<stdio.h>** header file.

Syntax:

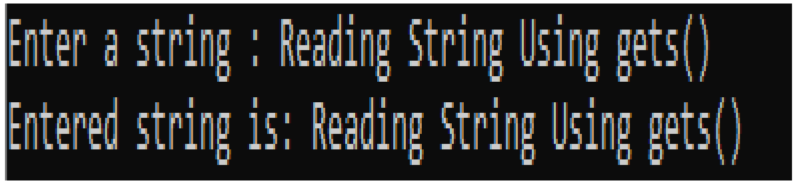
```
gets(str);
```

Example:

```
char line[80];
gets(line);
printf("%s",line);
```

Reading Strings from Terminal: Using gets()

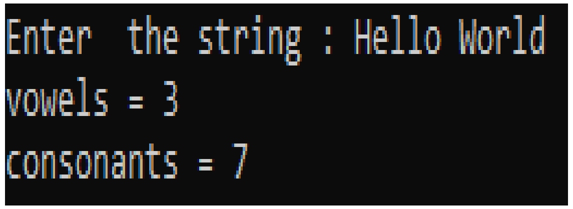
```
void main()
{
    char str[50];

    printf("Enter a string : ");
    gets(str);

    printf("Entered string is: %s", str);
}
```



```
Enter a string : Reading String Using gets()
Entered string is: Reading String Using gets()
```

Example:Count vowels and consonants in string

```c
void main()
{
    char s[1000];
    int i,vowels=0,consonants=0;
    printf("Enter  the string : ");
    gets(s);

    for(i=0;s[i];i++)
    {
        if((s[i]>=65 && s[i]<=90)|| (s[i]>=97 && s[i]<=122))
        {
            if(s[i]=='a'|| s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u'
               ||s[i]=='A'||s[i]=='E'||s[i]=='I'||s[i]=='O' ||s[i]=='U')
              vowels++;
            else
             consonants++;
        }
    }
    printf("vowels = %d\n",vowels);
    printf("consonants = %d\n",consonants);
}
```

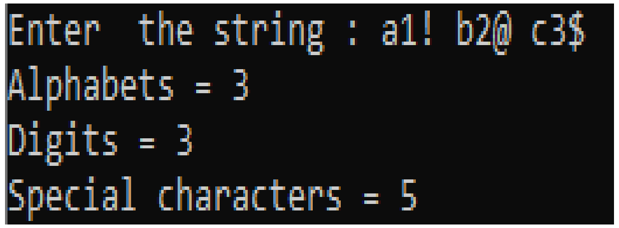


```
Enter  the string : Hello World
vowels = 3
consonants = 7
```

Practical 8.1

Take a user input for a string and calculate the number of alphabets, digits and special characters from the given input.

```c
void main()
{
    char s[1000];
    int i,alphabets=0,digits=0,specialcharacters=0;

    printf("Enter  the string : ");
    gets(s);

    for(i=0;s[i]!='\0';i++)
    {
        if((s[i]>='A' && s[i]<='Z')|| (s[i]>='a' && s[i]<='z') )
          alphabets++;
        else if(s[i]>='0' && s[i]<='9')
         digits++;
        else
         specialcharacters++;
    }
    printf("Alphabets = %d\n",alphabets);
    printf("Digits = %d\n",digits);
    printf("Special characters = %d", specialcharacters);
}
```



```
Enter  the string : a1! b2@ c3$
Alphabets = 3
Digits = 3
Special characters = 5
```

Reading Strings from Terminal

scanf() ends taking input upon encountering a whitespace, newline or EOF.

gets() considers a whitespace as a part of the input string and ends the input upon encountering newline or EOF.

getchar() is used to read a single character from a keyboard on the output screen. This character can be any character from the keyboard (a-z, A-Z, 0–9, !@#$%^&*() =+_{}:”?><,./;’[]\|….any character)

Reading Strings from Terminal

scanf	getchar
Read input from the standard input until encountering a whitespace, newline or EOF.	Read character only from the standard input stream(stdin) which is the keyboard
It takes the format string and variables with their addresses as parameters	It does not take any parameters
It reads data according to the format specifier	It reads a single character from the keyboard

Reading Strings from Terminal

Scanf	gets
when scanf() is used to read string input it stops reading when it encounters **whitespace, newline or End Of File**	when gets() is used to read input it stops reading input when it encounters **newline or End Of File.** It does not stop reading the input on encountering whitespace as it considers whitespace as a string.
It is used to read input of **any datatype**	It is used only for **string** input.
scanf() function takes **the format string and list of addresses of variables**	On other hand gets() function takes the **name of the variable** to store the received value.
Syntax: scanf("%s", name);	**Syntax:** gets(name);
scanf() function can read multiple values of different data types.	However on other hand gets() function will only get character string data.

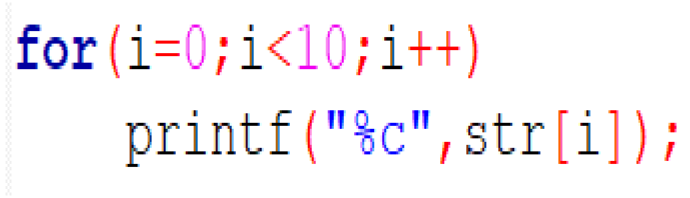
Writing Strings to Screens

1. Using **printf** function
2. Using **putchar** function(Used to print single character)
3. Using **puts** function

Writing Strings to Screens: Using printf

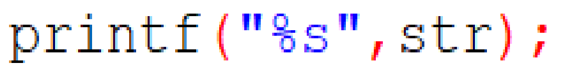
The familiar output function printf can be used with,

1) %c format specification (Print character by character)



```
for(i=0;i<10;i++)
    printf("%c",str[i]);
```

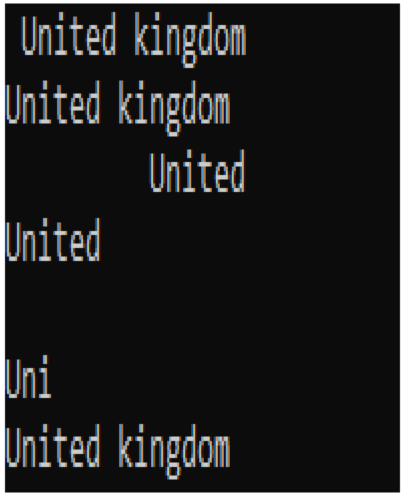
2)%s format specification (Print entire string)



```
printf("%s",str);
```

Writing Strings to Screens: Using printf

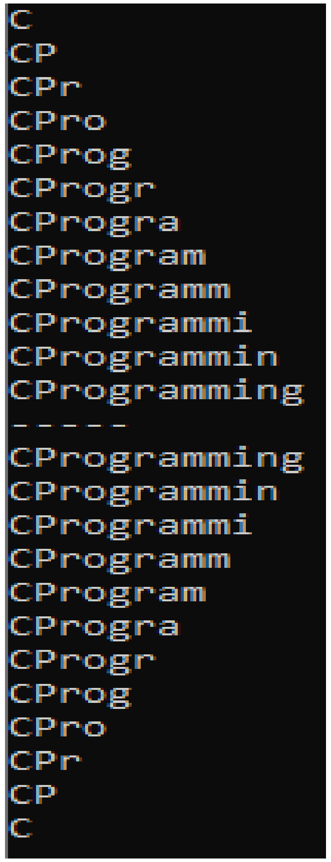
```c
void main()
{
char country[15]="United kingdom";
printf("%15s\n",country);
printf("%5s\n",country);
printf("%15.6s\n",country);
printf("%-15.6s\n",country);
printf("%15.0s\n",country);
printf("%.3s\n",country);
printf("%s\n",country);
}
```



```
 United kingdom
United kingdom
         United
United

Uni
United kingdom
```

Writing Strings to Screens: Using printf

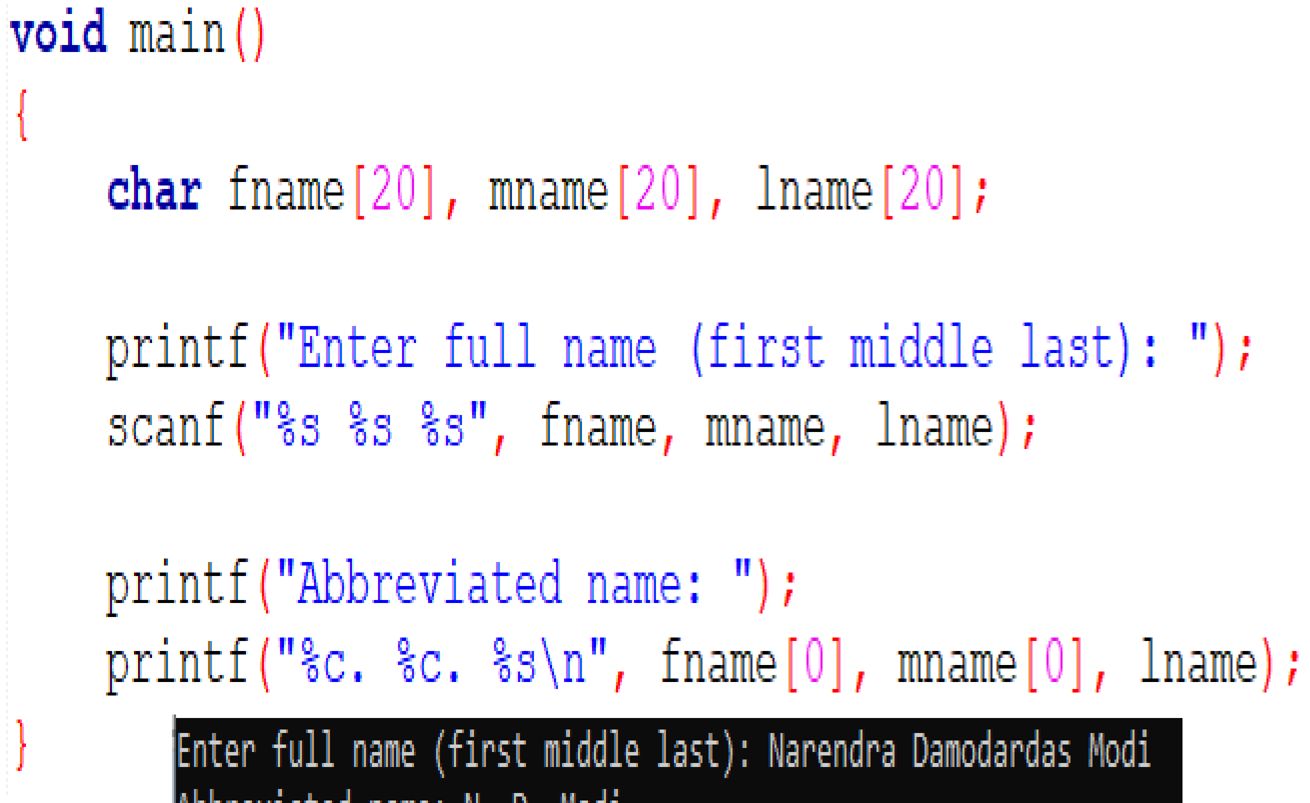
```c
void main()
{
      int c,d;
      char string[ ] = "CProgramming";
      for(c=0;c<=11;c++)
      {
            d=c+1;
            printf("%-12.*s\n",d, string);
      }
      printf("-----\n");

      for(c=11;c>=0;c--)
      {
            d=c+1;
            printf("%-12.*s\n",d, string);
      }
}
```



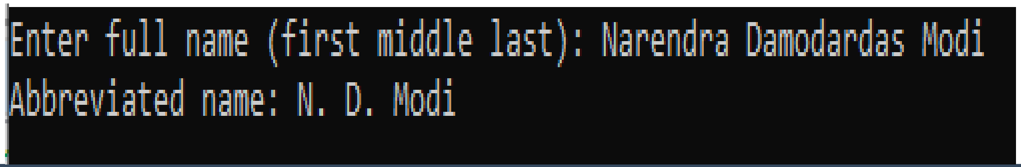
```
C
CP
CPr
CPro
CProg
CProgr
CProgra
CProgram
CProgramm
CProgrammi
CProgrammin
CProgramming
-----
CProgramming
CProgrammin
CProgrammi
CProgramm
CProgram
CProgra
CProgr
CProg
CPro
CPr
CP
C
```

Practical 8.2

Write a program that takes a set of names of individual and abbreviates the first, middle and other names except the last name by their first letter.



```c
void main()
{
    char fname[20], mname[20], lname[20];

    printf("Enter full name (first middle last): ");
    scanf("%s %s %s", fname, mname, lname);

    printf("Abbreviated name: ");
    printf("%c. %c. %s\n", fname[0], mname[0], lname);
}
```



```
Enter full name (first middle last): Narendra Damodardas Modi
Abbreviated name: N. D. Modi
```

Writing Strings to Screens: Using putchar

putchar() as the name states prints only one character at a time.

Syntax:

```
putchar('A');
```

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

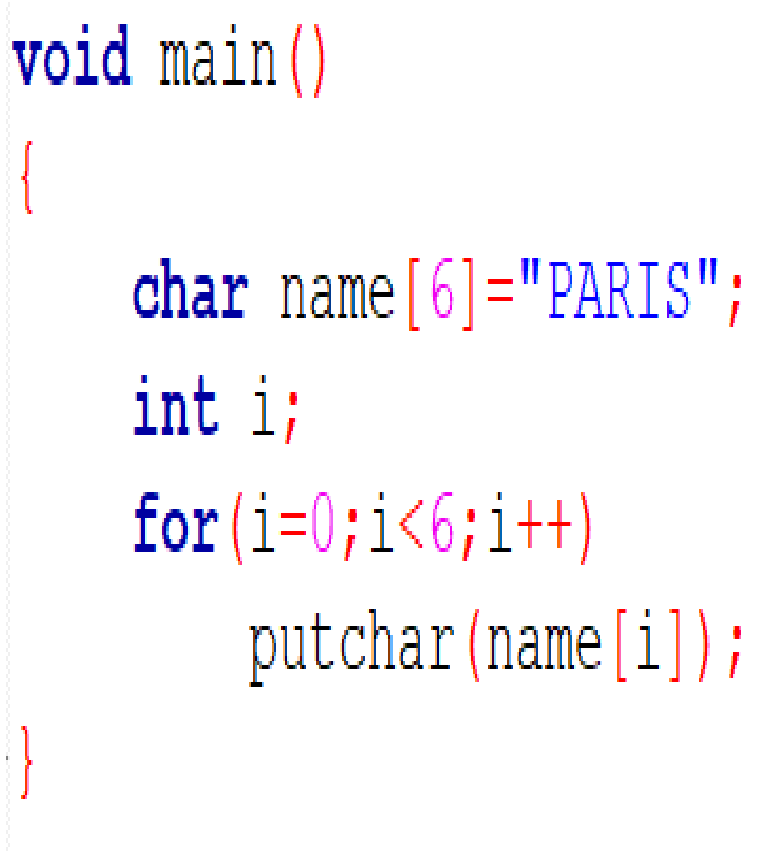
is equivalent to

```
char ch='A';
printf("%c",ch);
```

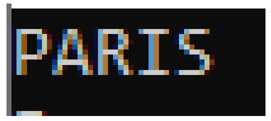
NOTE: putchar function requires one parameter.

Writing Strings to Screens: Using putchar

In order to print a string, we have to use **putchar** function repeatedly until a terminating character is encountered



```
void main()
{
    char name[6]="PARIS";
    int i;
    for(i=0;i<6;i++)
        putchar(name[i]);
}
```



```
PARIS
```

Writing Strings to Screens: Using puts

Another method of printing a string of text is to use **puts** function from **<stdio.h>** header file.

Syntax:

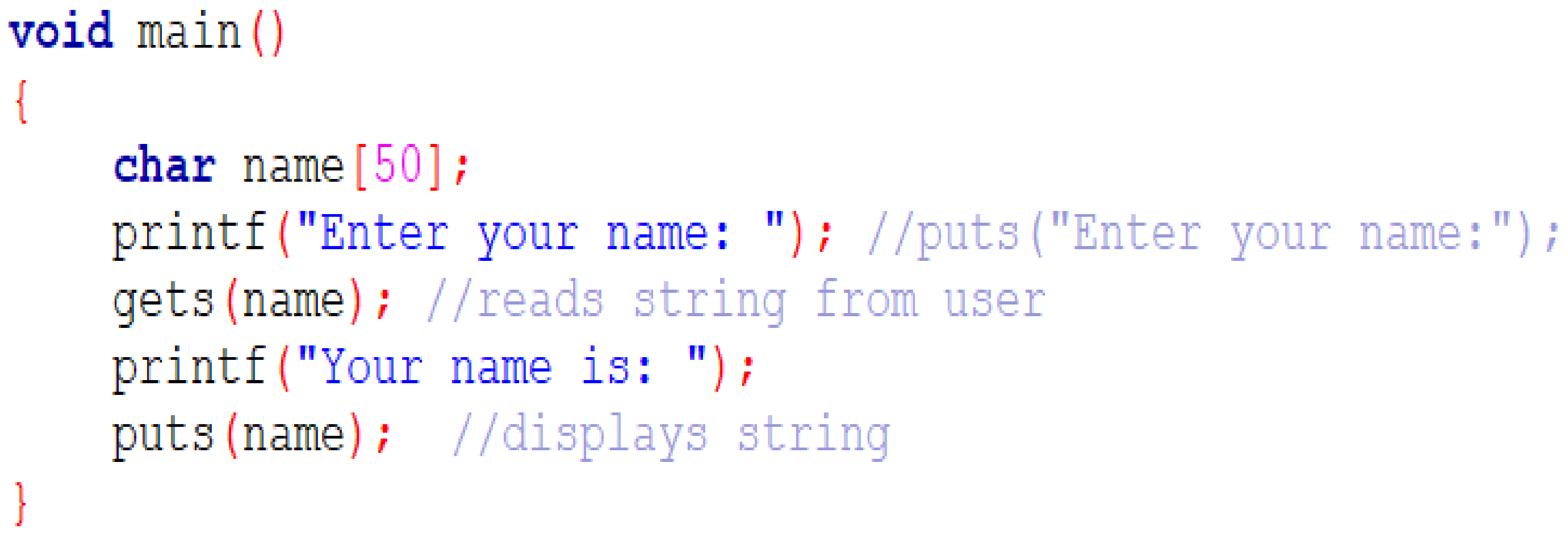
```
puts(str);
```

Example:

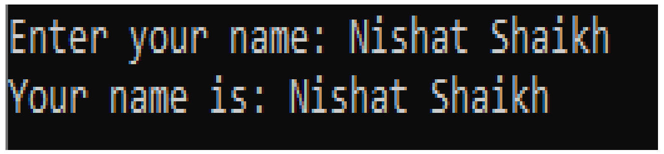
```
char line[80];
gets(line);
puts(line);
```

```
puts("Hello World");
```

Writing Strings to Screens: Using puts



```
void main()
{
    char name[50];
    printf("Enter your name: "); //puts("Enter your name:");
    gets(name); //reads string from user
    printf("Your name is: ");
    puts(name);  //displays string
}
```



```
Enter your name: Nishat Shaikh
Your name is: Nishat Shaikh
```

Writing Strings to Screens

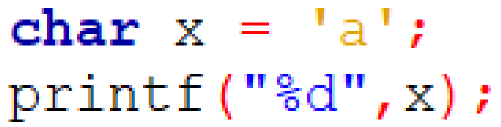
printf	Puts
C function to print a formatted string to the standard output stream which is the computer screen	C library function that writes a string to stdout or standard output
Syntax: printf(“%s”,str);	**Syntax:** puts(str);
Does not move the cursor to the next line, but the programmer can **use \n** to bring the cursor to the next line	Moves the cursor to the **next line**
The implementation of printf is **complicated** than puts	The implementation of puts is **simpler** than printf

1. Write a program which read a string and rewrite it in the alphabetical order. (i.e if Input : STRING then Output : GINRST)

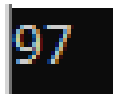
Arithmetic Operations On Characters

- C allows us to manipulate characters the same way we do with numbers.
- Whenever a character constant or character variable is used in an expression, it is automatically converted into integer value by the system.

For eg, if the machine uses the ASCII representation, then,



```c
char x = 'a';
printf("%d",x);
```



```
97
```

Arithmetic Operations On Characters

- It is also possible to perform **arithmetic operations** on the character constants and variables.

```
char x = 'z'-1;
printf("%d",x);
```

```
121
```

- We may also use character constants in **relational expressions**.

```
ch >= 'A' && ch <= 'Z'
```

It will test whether the character is an **upper-case letter**.

- We can convert a **character digit** to its equivalent **integer value**

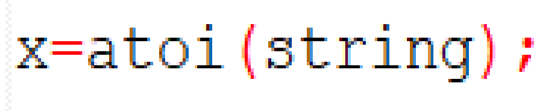
```
int x;
x='7'-'0';  //55-48=7
printf("%d",x);
```

```
7
```

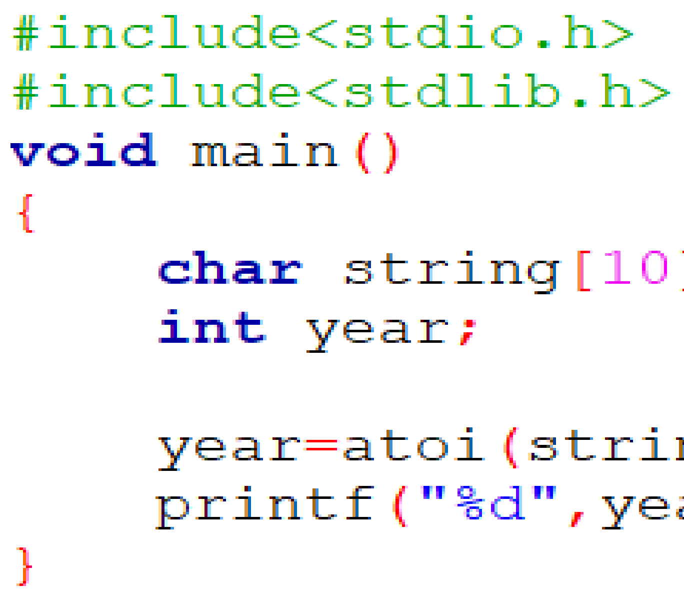
Arithmetic Operations On Characters

atoi: A function that converts a **string of digits** into their **integer values**

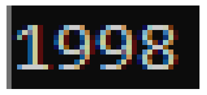
Syntax:



```
x=atoi(string);
```



```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    char string[10]="1998";
    int year;

    year=atoi(string);
    printf("%d",year);
}
```

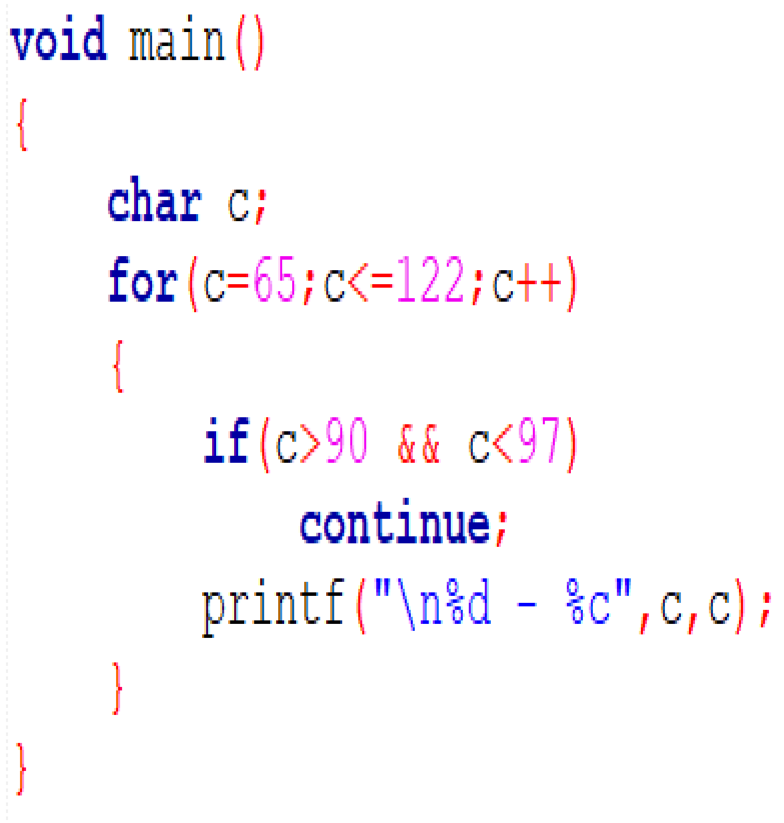


```
1998
```

Header file : stdlib.h

Arithmetic Operations On Characters

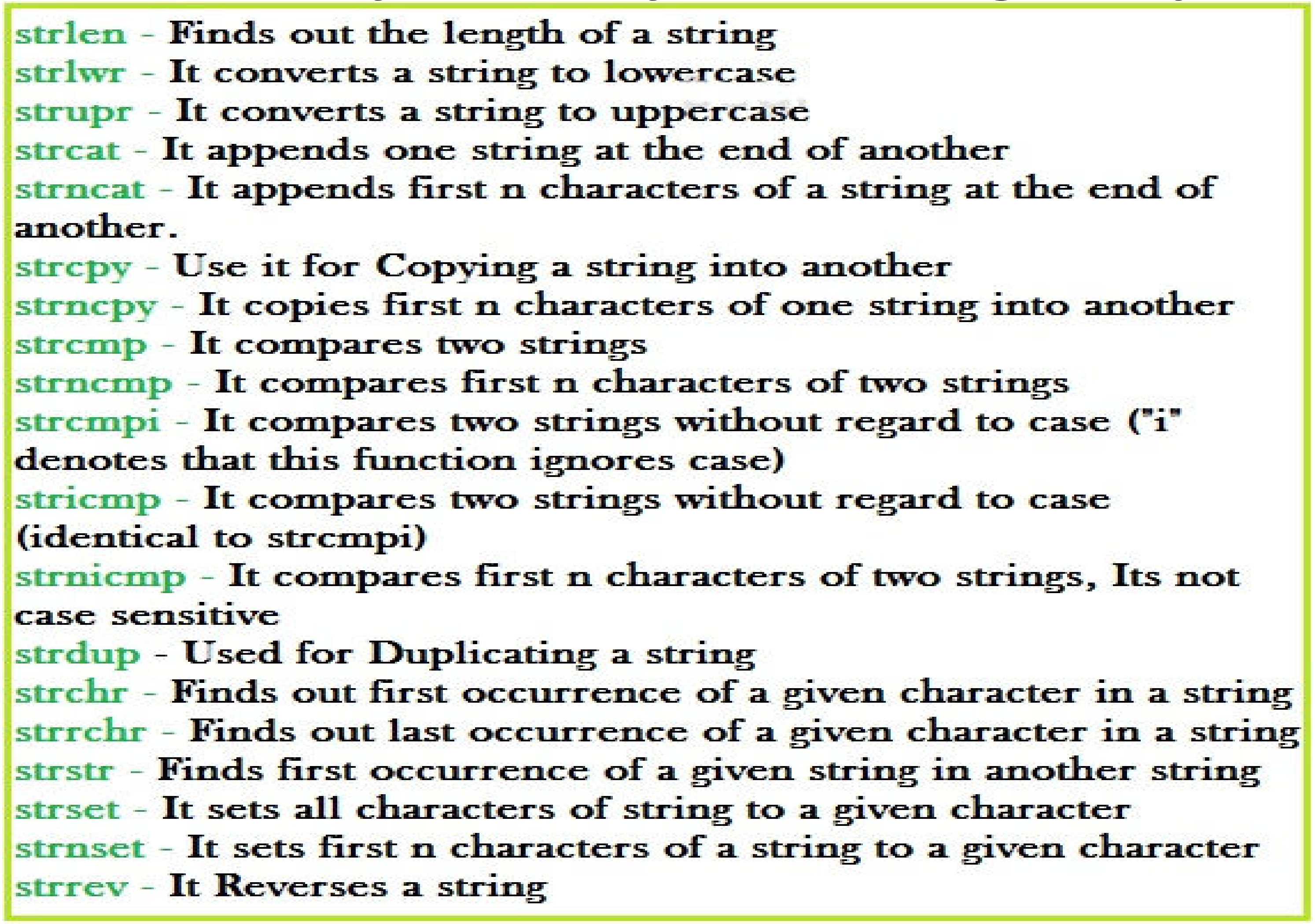
WAP to print the alphabet set a to z and A to Z in decimal and character form.



```
void main()
{
    char c;
    for(c=65;c<=122;c++)
    {
        if(c>90 && c<97)
            continue;
        printf("\n%d - %c",c,c);
    }
}
```

String handling functions

Used to carry out many of the string manipulations.



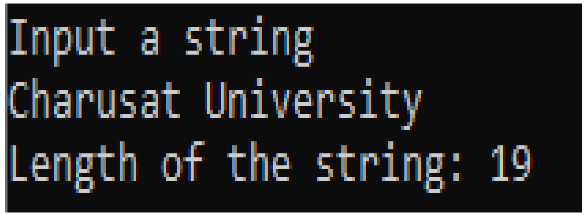
strlen - Finds out the length of a string
strlwr - It converts a string to lowercase
strupr - It converts a string to uppercase
strcat - It appends one string at the end of another
strncat - It appends first n characters of a string at the end of another.
strcpy - Use it for Copying a string into another
strncpy - It copies first n characters of one string into another
strcmp - It compares two strings
strncmp - It compares first n characters of two strings
strcmpi - It compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp - It compares two strings without regard to case (identical to strcmpi)
strnicmp - It compares first n characters of two strings, Its not case sensitive
strdup - Used for Duplicating a string
strchr - Finds out first occurrence of a given character in a string
strrchr - Finds out last occurrence of a given character in a string
strstr - Finds first occurrence of a given string in another string
strset - It sets all characters of string to a given character
strnset - It sets first n characters of a string to a given character
strrev - It Reverses a string

Header file :

#include<string.h>

Example: Calculate String length without strlen()

```c
void main()
{
  char s[1000];
  int count = 0;

  printf("Input a string\n");
  gets(s);

  while (s[count] != '\0')
    count++;

  printf("Length of the string: %d\n", count);
}
```



```
Input a string
Charusat University
Length of the string: 19
```

strlen() Function

strlen: counts and returns number of characters(length) of string.

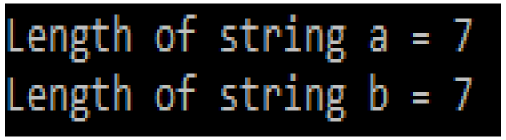
Syntax:

```
n=strlen(string);
```

NOTE: It stops counting the character when null character is found. Because, null character indicates the end of the string in C.

strlen() Function

```
#include <stdio.h>
#include <string.h>
void main()
{
    char a[20]="Program";
    char b[20]={'P','r','o','g','r','a','m','\0'};

    printf("Length of string a = %d \n",strlen(a));
    printf("Length of string b = %d \n",strlen(b));
}
```

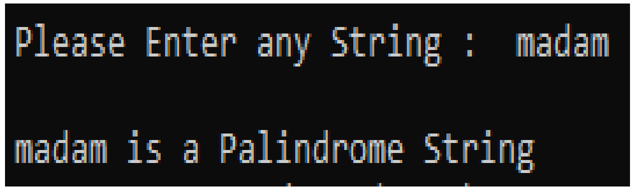


```
Length of string a = 7
Length of string b = 7
```

Practical 8.3

WAP to check if the user inputted string is palindrome or not.

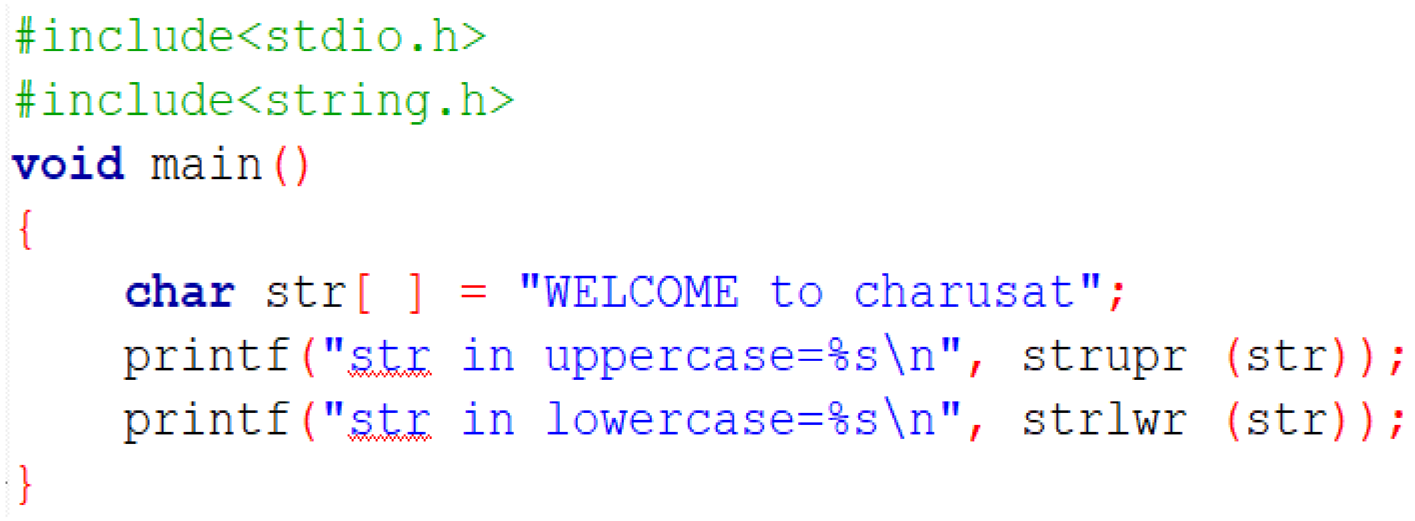
```c
#include <stdio.h>
#include <string.h>
void main()
{
    char str[100];
    int i, len, flag=0;

    printf("\n Please Enter any String :   ");
    gets(str);
    len = strlen(str);

    for(i = 0; i < len/2; i++)
    {
        if(str[i] != str[len - i - 1])
        {
            flag = 1;
            break;
        }
    }
    if(flag == 0)
    {
        printf("\n %s is a Palindrome String", str);
    }
    else
    {
        printf("\n %s is Not a Palindrome String", str);
    }
}
```



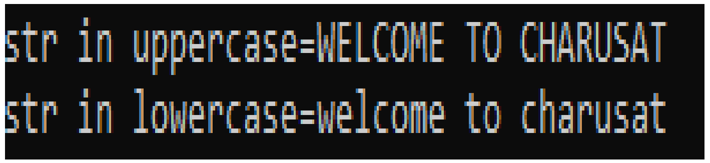
```
Please Enter any String :  madam

madam is a Palindrome String
```

strupr() & strlwr() Function

strupr: convert a given string into uppercase
strlwr: convert a given string into lowercase



```c
#include<stdio.h>
#include<string.h>
void main()
{
    char str[ ] = "WELCOME to charusat";
    printf("str in uppercase=%s\n", strupr (str));
    printf("str in lowercase=%s\n", strlwr (str));
}
```



```
str in uppercase=WELCOME TO CHARUSAT
str in lowercase=welcome to charusat
```

Putting Strings Together without strcat()

Just as we cannot assign one string to another directly, we cannot join two strings together by the simple arithmetic addition.

```
string3=string1+string2;
string2=string1+"hello";
```

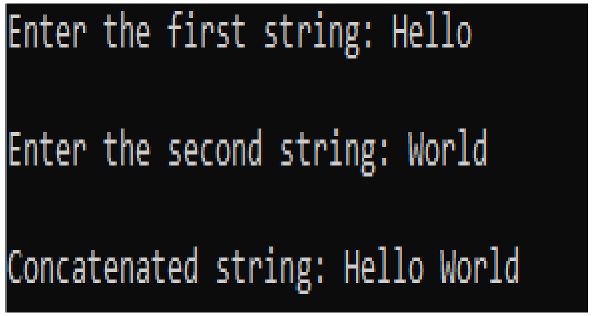
NOT VALID

- The characters from string1 and string2 should be copied into the string3 one after the other.
- The size of the array string3 should be large enough to hold the total characters.

The process of combining two strings together is called **concatenation**.

Example 3.1:Putting Strings Together without strcat()

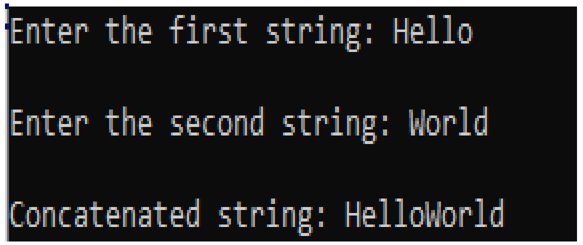
```
void main()
{
    char s1[20];
    char s2[20];
    char s3[50];
    int i,j;

    printf("Enter the first string: ");
    scanf("%s",s1);
    printf("\nEnter the second string: ");
    scanf("%s",s2);

    for(i=0;s1[i]!='\0';i++)
        s3[i]=s1[i];
    s3[i]=' ';

    for(j=0;s2[j]!='\0';j++)
        s3[i+j+1]=s2[j];
    s3[i+j+1]='\0';

    printf("\nConcatenated string: %s\n",s3);
}
```



```
Enter the first string: Hello

Enter the second string: World

Concatenated string: Hello World
```

Example 3.2:Putting Strings Together without strcat()

```c
void main()
{
    char s1[20];
    char s2[20];
    int i,j;

    printf("Enter the first string: ");
    scanf("%s",s1);
    printf("\nEnter the second string: ");
    scanf("%s",s2);

    for(i=0;s1[i]!='\0';i++);

    for(j=0;s2[j]!='\0';j++)
    {
        s1[i]=s2[j];
        i++;
    }
    s1[i]='\0';
   printf("\nConcatenated string: %s", s1);
}
```



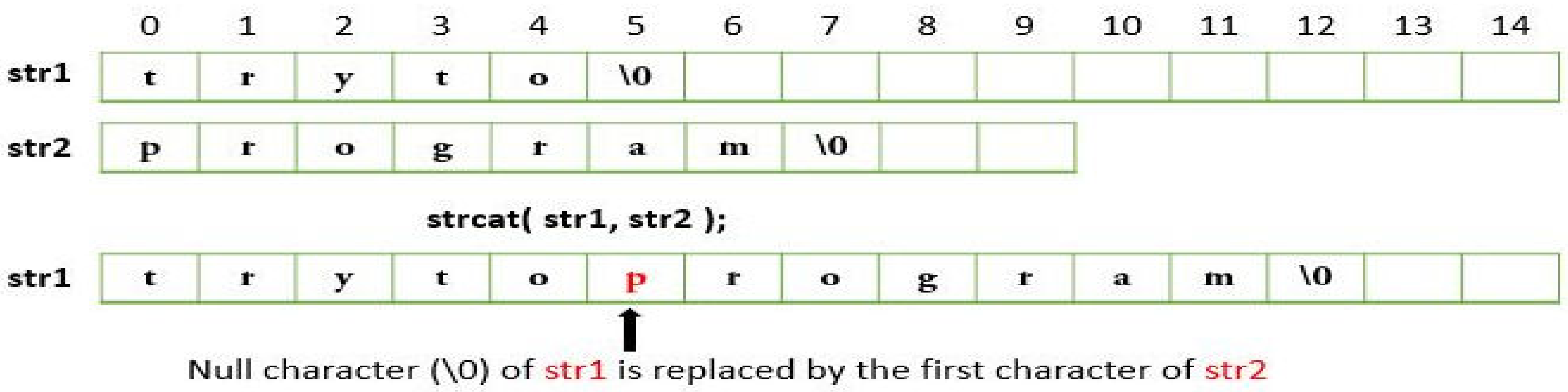
```
Enter the first string: Hello

Enter the second string: World

Concatenated string: HelloWorld
```

strcat() Function

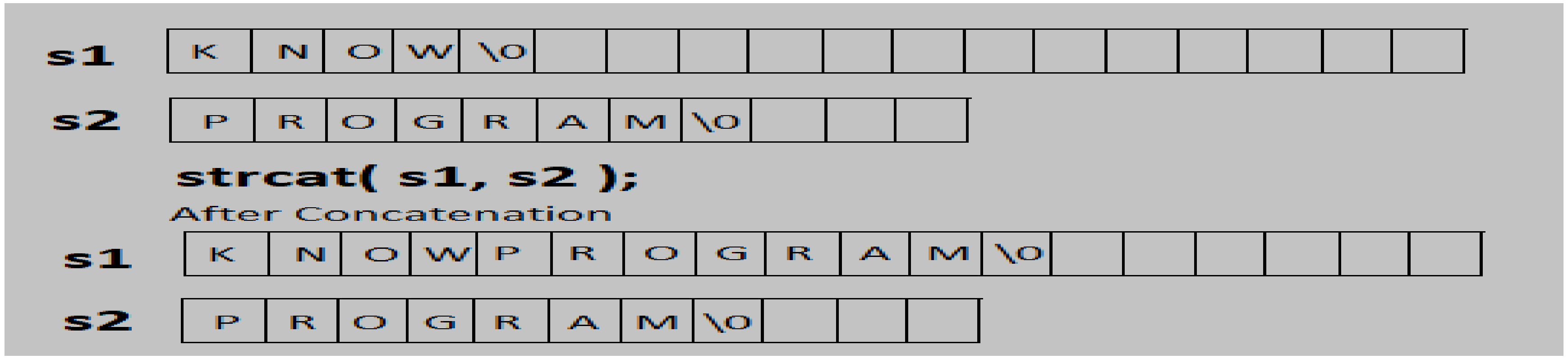
strcat: joins(concatenates) two strings together.

Syntax:

```
strcat(string1,string2); //string2 is appended to string1
```



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
str1 t r y t o \0
str2 p r o g r a m \0
strcat(str1, str2);
str1 t r y t o p r o g r a m \0
Null character (\0) of str1 is replaced by the first character of str2



strcat() Function

NOTE: Make sure that the **size** of string1(to which string2 is appended) is **large enough** to accommodate the final string.

- strcat function may also append a string constant to a string variable.

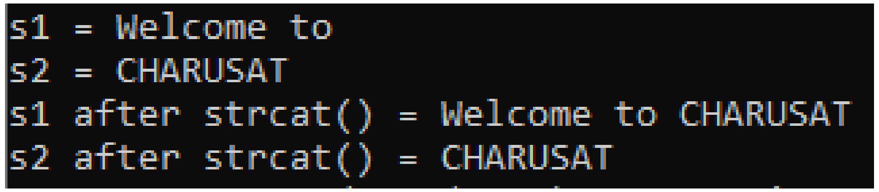
```
strcat(string1,"GOOD");
```

- C permits **nesting** of functions.

```
strcat(strcat(string1,string2),string3);
```

strcat() Function

```c
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[20]="Welcome to ";
    char s2[20]="CHARUSAT";

    printf("\ns1 = %s",s1);
    printf("\ns2 = %s",s2);

    strcat(s1,s2);

    printf("\ns1 after strcat() = %s",s1);
    printf("\ns2 after strcat() = %s",s2);
}
```



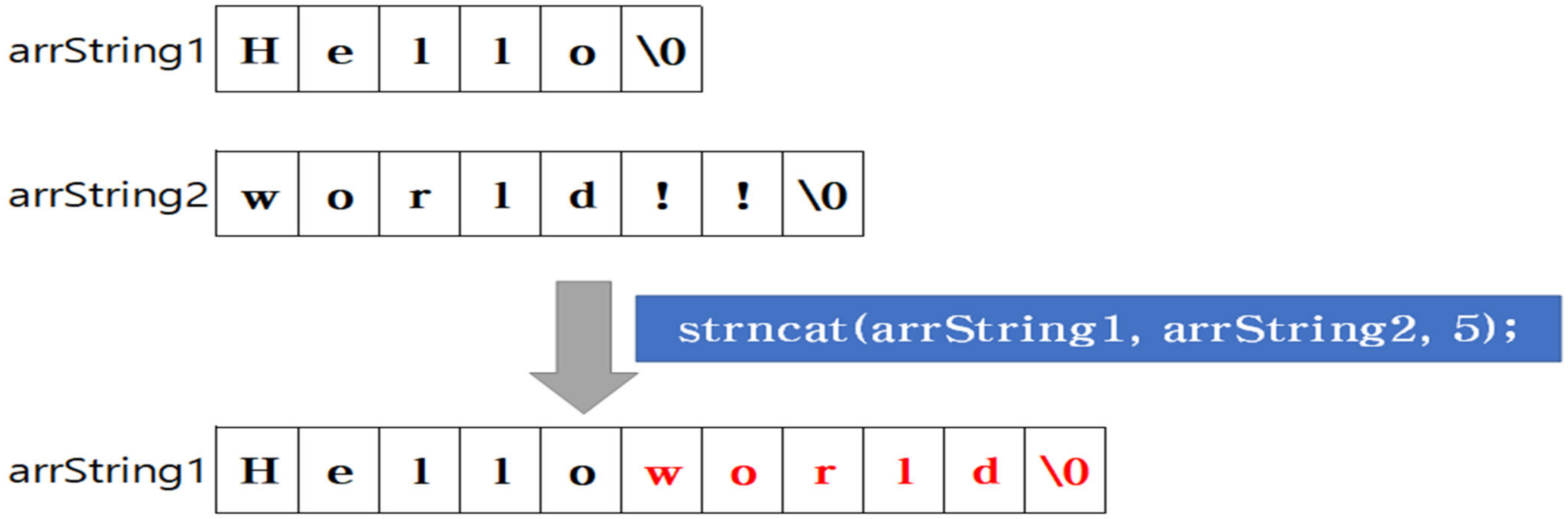
```
s1 = Welcome to
s2 = CHARUSAT
s1 after strcat() = Welcome to CHARUSAT
s2 after strcat() = CHARUSAT
```

strncat() Function

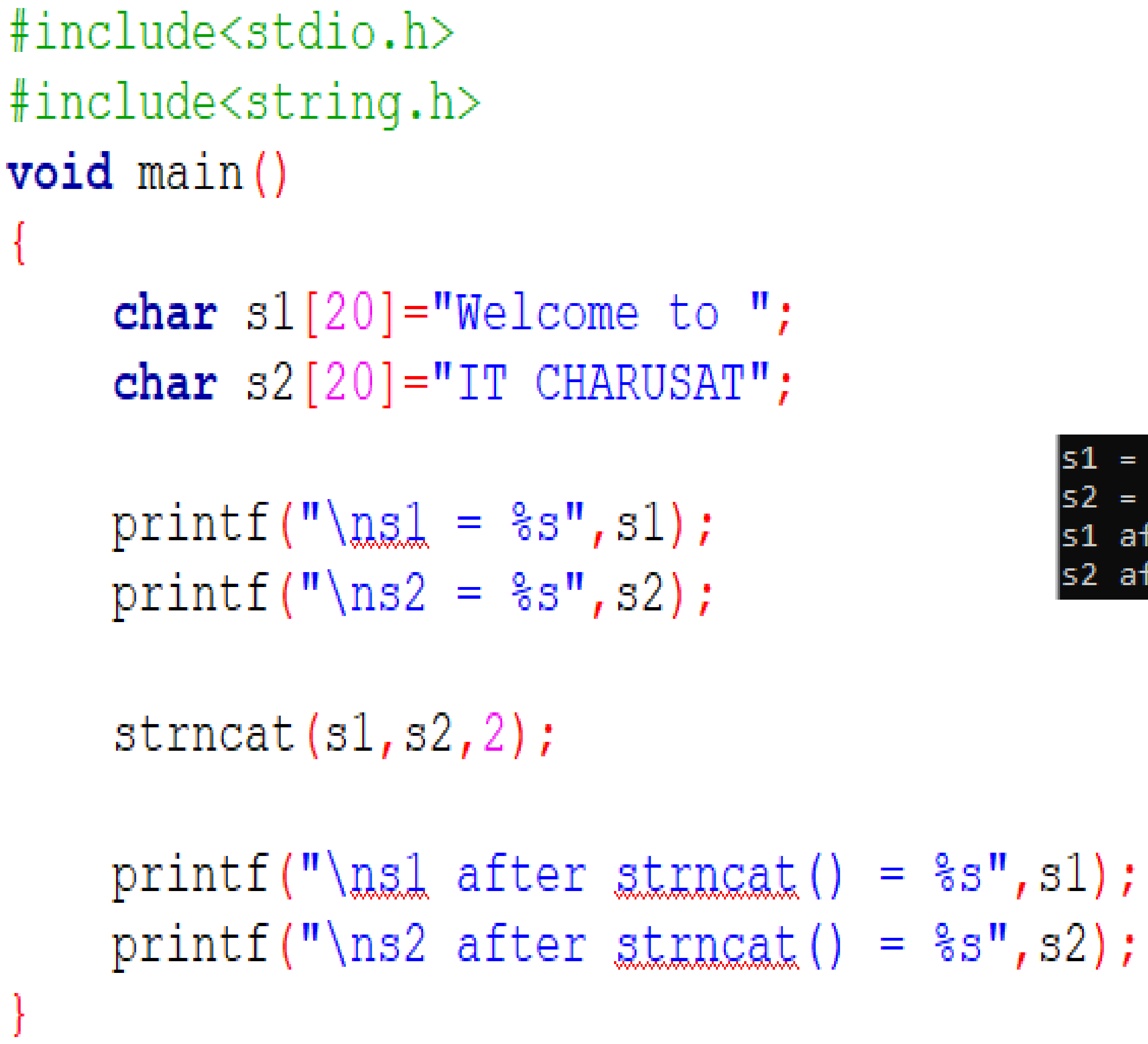
strncat: joins(concatenates) the left-most n characters of one string to the end of another.

Syntax:

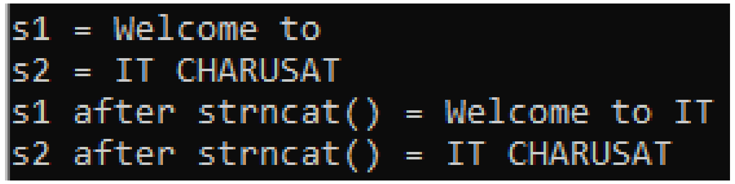
```
strncat(s1,s2,n);
```



strncat() Function



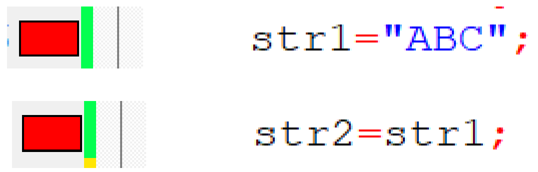
```c
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[20]="Welcome to ";
    char s2[20]="IT CHARUSAT";

    printf("\ns1 = %s",s1);
    printf("\ns2 = %s",s2);

    strncat(s1,s2,2);

    printf("\ns1 after strncat() = %s",s1);
    printf("\ns2 after strncat() = %s",s2);
}
```



```
s1 = Welcome to
s2 = IT CHARUSAT
s1 after strncat() = Welcome to IT
s2 after strncat() = IT CHARUSAT
```

Copy string without strcpy()

C does not provide operators that work on strings directly. For example, **we can not assign one string to another using assignment statement directly**

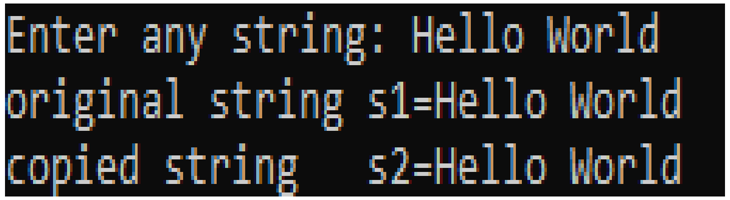


```
str1="ABC";
str2=str1;
```

To Copy One String To Another String, **we can do character-by-character.**

Example 1:Copy One String To Another String without strcpy()

```
void main()
{
     char s1[1000],s2[1000];
     int i;

     printf("Enter any string: ");
     gets(s1); //or scanf("%s",s1);
     for(i=0;s1[i]!='\0';i++) //or for(i=0;s1[i];i++)
     {
          s2[i]=s1[i];
     }
     s2[i]='\0';

     printf("original string s1=%s\n",s1);
     printf("copied string   s2=%s",s2);
 }
```



```
Enter any string: Hello World
original string s1=Hello World
copied string   s2=Hello World
```

strcpy() Function

strcpy: copies contents of one string into another string.

Syntax:

```
strcpy(string1,string2);
```

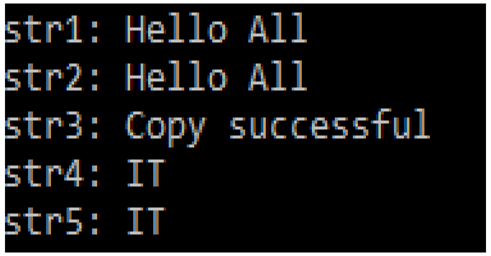
- string2 may be a **character array variable or a string constant**

```
strcpy(city1,city2);
strcpy(city1,"DELHI");
```

- **NOTE:** When you use strcpy(), the size of the destination string should be large enough to store the copied string. Otherwise, it may result in **undefined behavior**.

strcpy() Function

```c
#include<stdio.h>
#include<string.h>
void main ()
{
    char str1[40]="Hello All";
    char str2[40] = "CHARUSAT";
    char str3[40];
    char str4[40];
    char str5[40] = "IT";

    strcpy(str2, str1);
    strcpy(str3, "Copy successful");
    strcpy(str4, str5);

    printf ("str1: %s\nstr2: %s\nstr3: %s\nstr4: %s\nstr5: %s", str1, str2, str3, str4,str5);
}
```



```
str1: Hello All
str2: Hello All
str3: Copy successful
str4: IT
str5: IT
```

strncpy() Function

strncpy: copies left-most n characters of one string into another string.

Syntax:

```
strncpy(s1,s2,n);
```

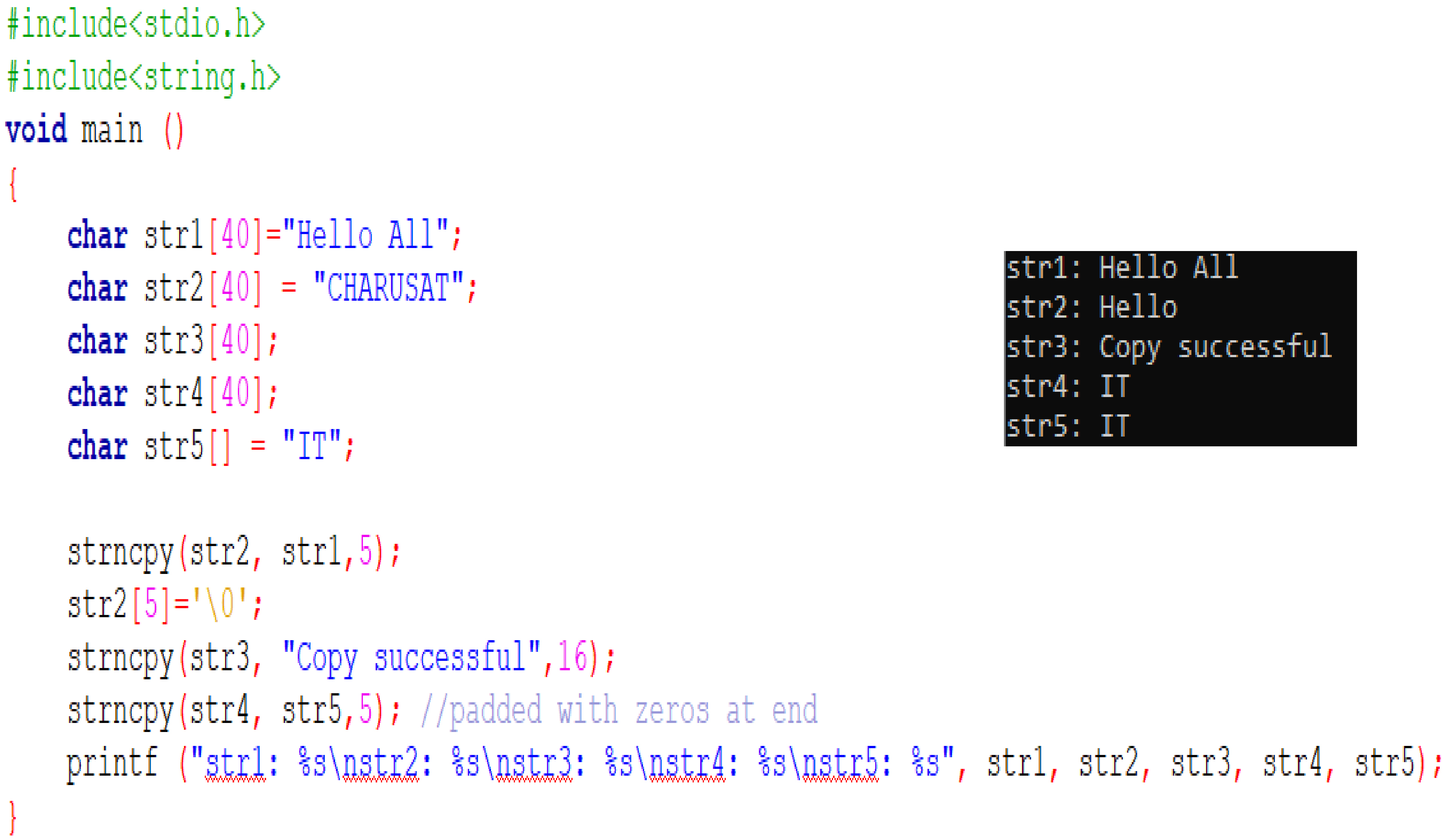
Example:

```
strncpy(s1,s2,5);
```

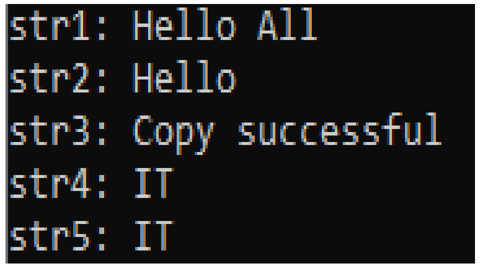
- It copies the **first 5 characters** of the string s2 into string s1.
- Since the first 5 characters may not include the **terminating null character**, we have to **place it explicitly** in the 6th position of s1

```
s1[6]='\0';
```

strncpy() Function



```c
#include<stdio.h>
#include<string.h>
void main ()
{
    char str1[40]="Hello All";
    char str2[40] = "CHARUSAT";
    char str3[40];
    char str4[40];
    char str5[] = "IT";

    strncpy(str2, str1,5);
    str2[5]='\0';
    strncpy(str3, "Copy successful",16);
    strncpy(str4, str5,5); //padded with zeros at end
    printf ("str1: %s\nstr2: %s\nstr3: %s\nstr4: %s\nstr5: %s", str1, str2, str3, str4, str5);
}
```



```
str1: Hello All
str2: Hello
str3: Copy successful
str4: IT
str5: IT
```

Comparison of Two Strings without strcmp()

Once again, C does **not permit the comparison of two strings directly.**

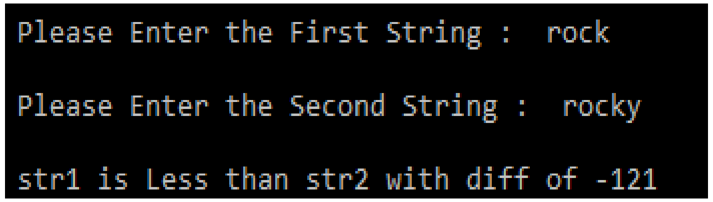
```
if(string1==string2)
if(string=="ABC")
```

NOT VALID

- Comparison of two strings **will be done character by character**
- The comparison is done until there is a **mismatch** or one of the **strings terminates into a null character**, whichever occurs first

Comparison of Two Strings without strcmp()

```
void main()
{
     char str1[100], str2[100];
     int diff, i;

     printf("\n Please Enter the First String :  ");
     gets(str1);
     printf("\n Please Enter the Second String :  ");
     gets(str2);

     for(i = 0; str1[i] != '\0' ||  str2[i]!= '\0' ; i++)
     {
          diff=(str1[i]-str2[i]);
          if(diff!=0)
               break;
     }

     if(diff<0)
     {
          printf("\n str1 is Less than str2 with diff of %d",diff);
     }
     else if(diff>0)
     {
          printf("\n str2 is Less than str1 with diff of %d",diff);
     }
     else
     {
          printf("\n str1 is Equal to str2 (diff=%d)",diff);
     }

}
```



```
Please Enter the First String :  rock

Please Enter the Second String :  rocky

str1 is Less than str2 with diff of -121
```

strcmp() Function

strcmp: compares two strings.

Syntax:

```
strcmp(string1,string2);
```

string1 and string2 may be string variables or string constants

```
strcmp(name1,name2);
strcmp(name1,"John");
strcmp("Rom","Ram");
```

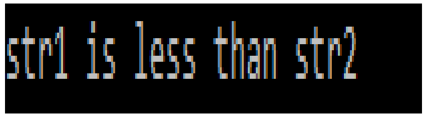
Return Value from strcmp()

Return Value	Remarks
0	if both the strings are equal
negative	if the ASCII value of first unmatched character of string str1 is **less than** the character in string str2
positive integer	if the ASCII value of first unmatched character of string str1 is **greater than** the character in string str2

NOTE: strcmp( ) function is case sensitive. i.e, “A” and “a” are treated as different characters.

strcmp() Function

```c
#include <stdio.h>
#include<string.h>
void main()
{
   char str1[15]="their";
   char str2[15]="there";
   int diff;

   diff = strcmp(str1, str2);
   //printf("%d\n",diff);

   if(diff < 0)
        printf("str1 is less than str2");
   else if(diff > 0)
        printf("str2 is less than str1");
   else
        printf("str1 is equal to str2");
}
```



```
str1 is less than str2
```

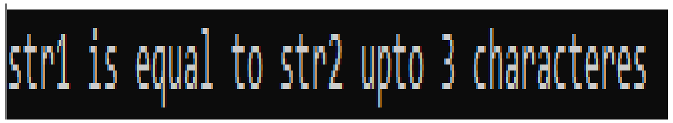
strncmp() Function

strncmp: compares the left-most n characters of two strings.

Syntax:

```
strncmp(str1,str2,n);
```

```
#include <stdio.h>
#include <string.h>
void main ()
{
    char str1[15]="their";
    char str2[15]="there";
    int diff;

    diff=strncmp(str1,str2,3);

    if(diff < 0)
       printf("str1 is less than str2");
    else if(diff > 0)
       printf("str2 is less than str1");
    else
       printf("str1 is equal to str2 upto 3 characteres");
}
```



```
str1 is equal to str2 upto 3 characteres
```

strchr() & strrchr()Function

strchr: locate the first occurrence of specified character in given string

Syntax:

```
strchr(s1,'m');
```

strrchr: locate the last occurrence of specified character in given string

Syntax:

```
strrchr(s1,'m');
```

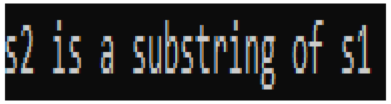
strstr() Function

strstr: locate substring in a string

Syntax:

```c
strstr(s1,s2);
strstr(s1,"ABC");
```

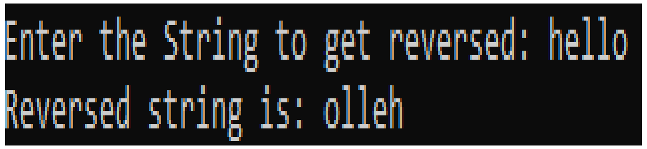
```c
#include <stdio.h>
#include <string.h>
void main()
{
    char s1[20] = "abcde";
    char s2[10] = "cd";

    if(strstr(s1, s2)==NULL)
        printf("substring is not found");
    else
        printf("s2 is a substring of s1");
}
```



```
s2 is a substring of s1
```

Example :Reverse string without strrev()

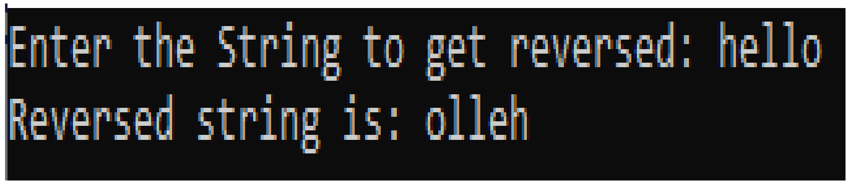
```c
void main()
{
    int i,n;
    char str[20];
    printf("Enter the String to get reversed: ");
    gets(str);
    n=strlen(str);
    printf("Reversed string is: ");
    for(i=n-1;i>=0;i--)
    {
       printf("%c",str[i]);
    }
}
```



```
Enter the String to get reversed: hello
Reversed string is: olleh
```

Example :Reverse string without strrev()

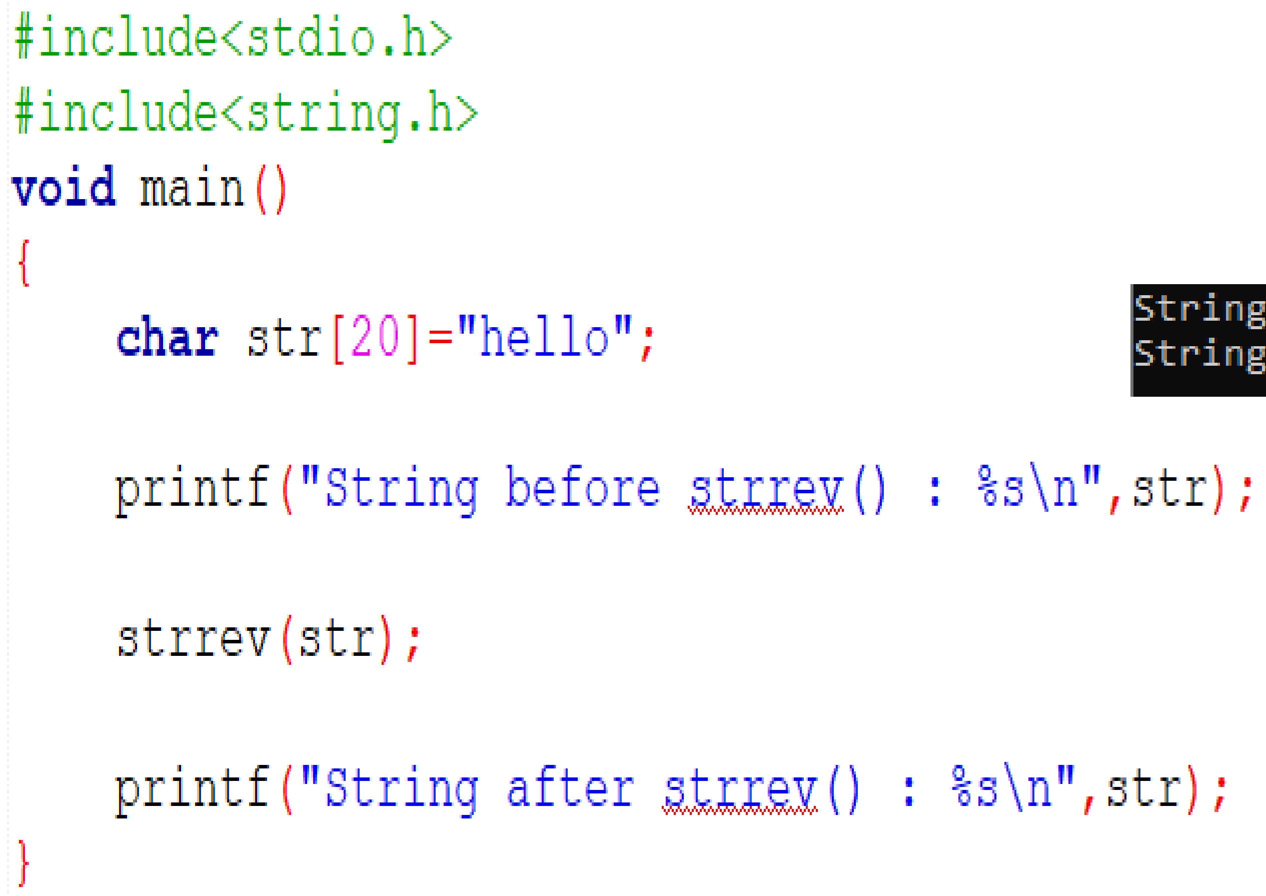
```c
void main()
{
    char s[1000], r[1000];
    int begin, end, count = 0;
    printf("Enter the String to get reversed: ");
    gets(s);

    // Calculating string length
    while (s[count] != '\0')
        count++;

    end = count - 1;

    for (begin = 0; begin < count; begin++)
    {
        r[begin] = s[end];
        end--;
    }
    r[begin] = '\0';
    printf("Reversed string is: %s\n", r);
}
```



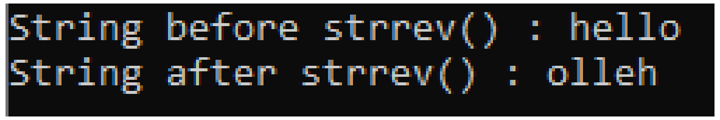
```
Enter the String to get reversed: hello
Reversed string is: olleh
```

strrev() Function

strrev: reverses a given string.



```c
#include<stdio.h>
#include<string.h>
void main()
{
    char str[20]="hello";

    printf("String before strrev() : %s\n",str);

    strrev(str);

    printf("String after strrev() : %s\n",str);
}
```



```
String before strrev() : hello
String after strrev() : olleh
```

1. s1, s2, and s3 are three string variables. Write a program to read two string constants into s1 and s2 and compare whether they are equal or not. If they are not, join them together. Then copy the contents of s1 to the variable s3. At the end, the program should print the contents of all the three variables and their lengths.
2. Write a program to find whether two given strings are permutations/anagrams of each other. (**HINT:** A Permutation/anagrams of a string is another string that contains same characters, only the order of characters can be different. For example, “abcd” and “dabc” are Permutation of each other)
3. Write a program to copy the last n characters of a character array in another character array. Also convert the lower case letters into upper case letters while copying.

Two dimensional character array (table of strings)

- **A list of names** can be treated as a **table of strings** and a **two-dimensional character array** can be used to store the entire list.

- For example, A character array **student[30][15]** may be used to store a list of 30 names each of length not more than 15 characters.

Two dimensional character array (table of strings)

Table of **five cities** cab be stored in a character array **city** using following declaration:

C	H	A	N	D	I	G	A	D	H
M	A	D	R	A	S				
H	Y	D	R	A	B	A	D		
A	H	M	E	D	A	B	A	D	
M	U	M	B	A	I				

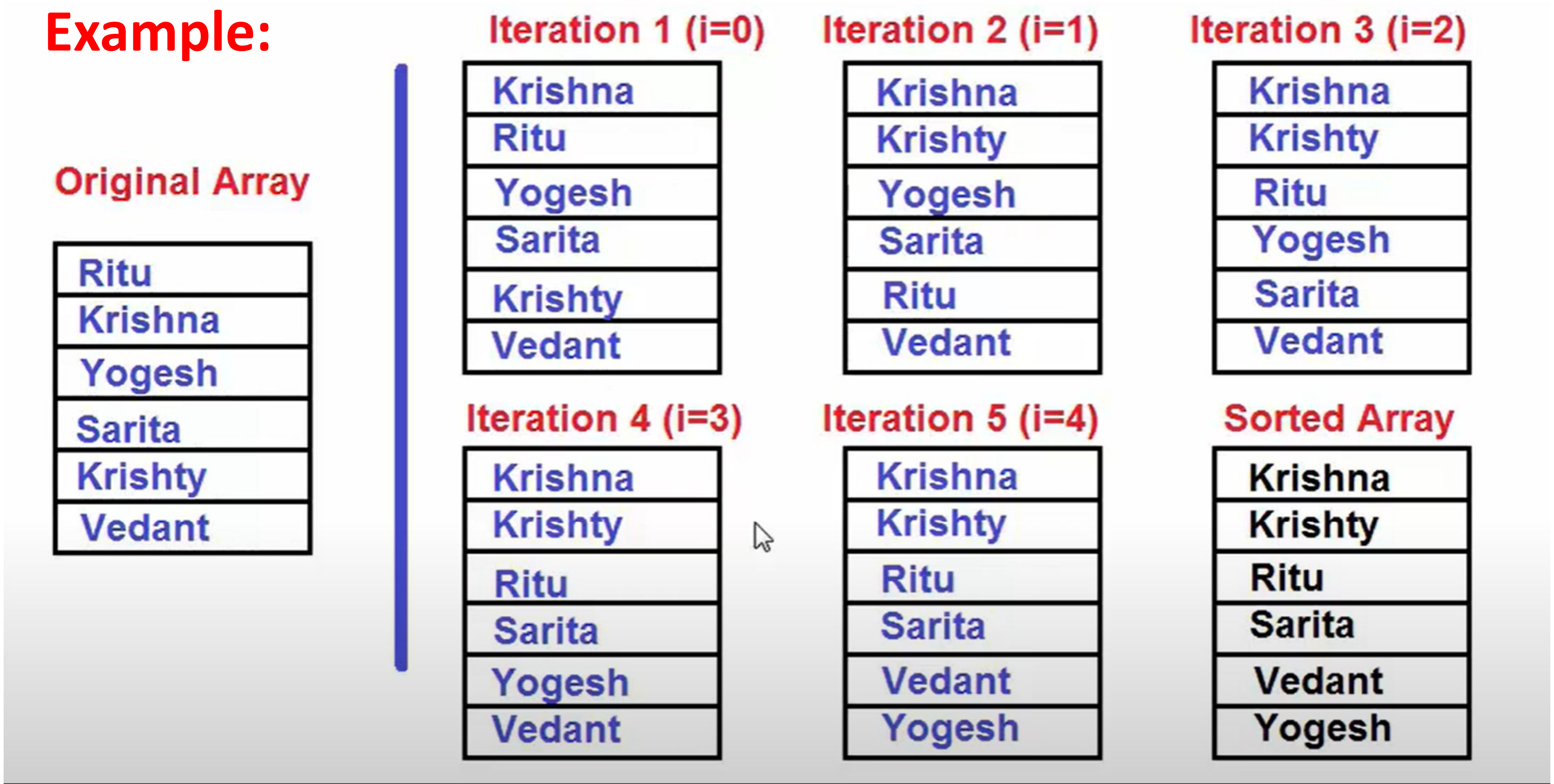
```
char city[ ][ ]
{
        "CHANDIGADH“,
        "MADRAS“,
        "HYDRABAD“,
        "AHMEDABAD“,
        "MUMBAI"
};
```

To access the name of the ith city, we write city[i-1].

Two dimensional character array (table of strings)

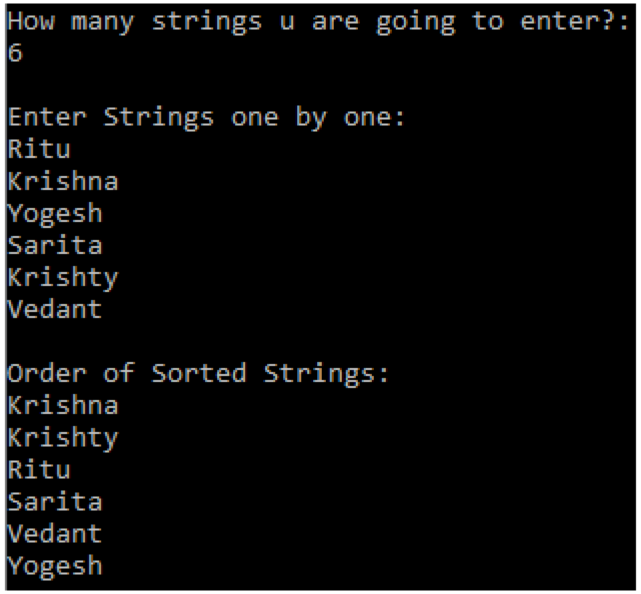
Write a program that would sort a list of names in alphabetical order.

Example:



Original Array	Iteration 1 (i=0)	Iteration 2 (i=1)	Iteration 3 (i=2)	Iteration 4 (i=3)	Iteration 5 (i=4)	Sorted Array
Ritu	Krishna	Krishna	Krishna	Krishna	Krishna	Krishna
Krishna	Ritu	Krishty	Krishty	Krishty	Krishty	Krishty
Yogesh	Yogesh	Yogesh	Ritu	Ritu	Ritu	Ritu
Sarita	Sarita	Sarita	Yogesh	Sarita	Sarita	Sarita
Krishty	Krishty	Ritu	Sarita	Yogesh	Vedant	Vedant
Vedant	Vedant	Vedant	Vedant	Vedant	Yogesh	Yogesh

```c
void main()
{
    int i,j,n;
    char str[25][25],temp[25];
    puts("How many strings u are going to enter?: ");
    scanf("%d",&n);

    puts("\nEnter Strings one by one: ");
    for(i=0;i<n;i++)
        scanf("%s",str[i]);

    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(strcmp(str[i],str[j])>0)
             {
                    strcpy(temp,str[i]);
                    strcpy(str[i],str[j]);
                    strcpy(str[j],temp);
             }
         }
    }
    printf("\nOrder of Sorted Strings:\n");
    for(i=0;i<n;i++)
        puts(str[i]);
}
```



```
How many strings u are going to enter?:
6

Enter Strings one by one:
Ritu
Krishna
Yogesh
Sarita
Krishty
Vedant

Order of Sorted Strings:
Krishna
Krishty
Ritu
Sarita
Vedant
Yogesh
```

Other Features of Strings

- Manipulating strings using **pointers**
- Using string as **function parameters**
- Declaring and defining strings as **members of structures**

End of Unit-08