

TCS201: Programming for Problem Solving

Unit-1

Multi-Dimensional Array and Strings

ARRAY

- Array is a collection of elements having same data type, same name and stored sequentially in a contiguous memory locations.
- Any element can be accessed by using
 - name of the array and
 - location of element in the array
- Arrays are of two types:
 - 1) Single dimensional array
 - 2) Multi-dimensional array

MULTI DIMENSIONAL ARRAYS:

Arrays with more than one dimension are called multi-dimensional arrays.

For ex:

int mat[3][3]; // declares a two dimensional array of size three rows and three columns

The above declaration inside the computer memory can be represented as follows:

	Column-1	Column-2	Column-3
Row-1	mat[0][0]	mat[0][1]	mat[0][2]
Row-2	mat[1][0]	mat[1][1]	mat[1][2]
Row-3	mat[2][0]	mat[2][1]	mat[2][2]

- Two dimensional array's are arranged in rows and columns form in the form of a matrix.
- The elements of the array index start with 0,0.
for ex. in the above figure the first element of integer type is stored at row=0 and col=0 index i.e at mat[0][0] and the second element is stored at row 0 and col=1 i.e at mat[0][1] & so on.

The General syntax of the 2D array is as follows:

data_type array_name[size_of_rows][size_of_cols];

where, data_type could be any basic types such as int,char or float etc.

array_name: any valid identifier name

size_of_rows: number of rows in 2D array

size_of_cols :number of columns in 2D array

Initialization of Two Dimensional Arrays At Compile Time:

For ex:

```
int matrix[3][3]= {{1,2,3},{4,5,6},{7,8,9}};
```

or

```
int matrix[3][3]= {1,2,3,4,5,6,7,8,9};
```

The above code can be represented as shown below:

	Col-1	Col-2	Col-3
Row-1	1	2	3
Row-2	4	5	6
Row-3	7	8	9

Example: Compile Time Initialization:

```
#include<stdio.h>
int main()
{
    int matrix[3][3]={ 1,2,3,4,5,6,7,8,9};
    int i,j;

    printf("\nThe elements of the matrix are:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d", matrix[i][j]);

        }
    }
}
```

Example: Run Time Initialization:

```
#include<stdio.h>
int main()
{
    int matrix[3][3];
    scanf("%d", &matrix[0][0]); //reads values entered into matrix at row=0 col=0
    scanf("%d", &matrix[0][1]); //reads values entered into row=0 and col=1.
    scanf("%d", &matrix[0][2]); //reads values entered into row=0 and col=2
    scanf("%d", &matrix[1][0]);
    scanf("%d", &matrix[1][1]);
    scanf("%d", &matrix[1][2]);
    scanf("%d", &matrix[2][0]);
    scanf("%d", &matrix[2][1]);
    scanf("%d", &matrix[2][2]);

}
```

The following loop reads 9 values entered by the user:

```
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        scanf("%d", &matrix[i][j]);
    }
}
```

The following loop displays 9 values to the output screen:

```
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        printf("%d ", matrix[i][j]);
    }
}
```

Example:

```
#include<stdio.h>
int main()
{
    int matrix[3][3];
    int i,j;
    printf("\nEnter the elements of the matrix of order [3x3]:\n");

    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("\nThe elements of the matrix are:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%2d", matrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

Enter the elements of the matrix of order[3x3]: 11 22 33 44 55 66 77 88 99

The elements of the matrix are:

11 22 33

44 55 66

77 88 99

Program : Write a C program to add the elements of square matrices. Display the resultant matrix to the output screen.

```
#include<stdio.h>
int main()
{
int matrix1[3][3], matrix2[3][3], matrix3[3][3];
int rows1, cols1, rows2, cols2;
printf("Enter the number of rows and columns of first matrix:");
scanf("%d%d",&rows1,&cols1);
printf("Enter the number of rows and columns of second matrix:");
scanf("%d%d",&rows2,&cols2);
if(rows1!=rows2 && cols1!=cols2)
{
    printf("Not a square matrix. Can't add elements!!!");
    exit(1);
}
printf("Enter the elements of first matrix:\n");
for(i=0;i<rows1;i++)
    for(j=0;j<cols1;j++)
    {
        scanf("%d", &matrix1[i][j]);
    }

printf("Enter the elements of second matrix:\n");
for(i=0;i< rows1;i++)
    for(j=0;j< cols;j++)
    {
        scanf("%d", &matrix2[i][j]);

    }

printf("The elements of first matrix:\n");
for(i=0;i< rows1;i++)
{
    for(j=0;j< cols;j++)
    {
        printf("%3d", matrix1[i][j]);
    }
    printf("\n");
}
printf("\nThe elements of second matrix:\n");

for(i=0;i< rows;i++)
{
    for(j=0;j< cols;j++)
    {
        printf("%3d", matrix2[i][j]);
    }
}
```

```

        printf("\n");
    }
    for(i=0;i< rows;i++)
    {
        for(j=0;j< cols;j++)
        {
            matrix3[i][j]= matrix1[i][j]+ matrix2[i][j];
        }
    }
    printf("\nThe elements of resultant matrix:\n");

    for(i=0;i< rows;i++)
    {
        for(j=0;j< cols;j++)
        {
            printf("%3d", matrix3[i][j]);
        }
        printf("\n");
    }
}

```

Output:

Enter the number of rows and columns of first matrix: 2 2

Enter the number of rows and columns of second matrix: 2 2

The elements of first matrix: 1 1 1 1

The elements of second matrix: 1 2 3 4

The elements of resultant matrix:

2 3

4 5

ROW- AND COLUMN-MAJOR ORDER REPRESENTATION IN MEMORY

- Row-major order and Column-major order are methods for storing multidimensional arrays in linear storage such as random access memory.
- In row-major order, the consecutive elements of a row reside next to each other, whereas the same holds true for consecutive elements of a column in column-major order.
- For example, the array could be stored in two possible ways:

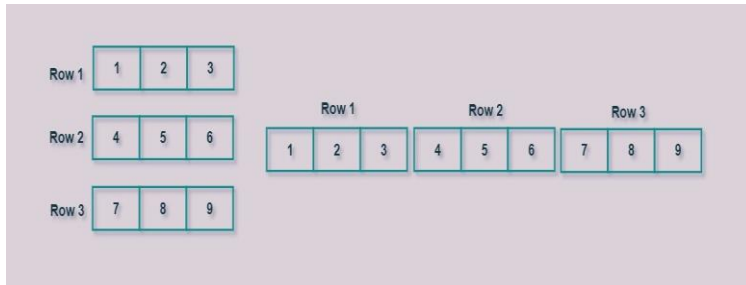
$A[2][3] = \begin{matrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{matrix}$

Address	Row-Major Order	Column Major Order
0	a_{00}	a_{00}
1	a_{01}	a_{10}
2	a_{02}	a_{01}
3	a_{10}	a_{11}
4	a_{11}	a_{02}
5	a_{12}	a_{12}

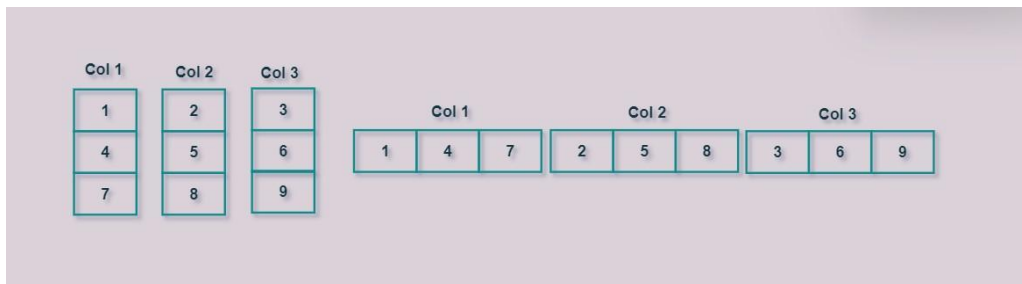
Example: Given matrix is:

	Col 1	Col 2	Col 3
Row 1	1	2	3
Row 2	4	5	6
Row 3	7	8	9

For the Row-wise traversal, we need to print row 1 first, row 2, and row 3, as shown below.



For the Column-wise traversal, you need to print col1 first, then col 2, and then col 3, as shown below.



Address Calculation in 2-d Array:

The 2-dimensional array can be defined as an array of arrays. The 2-Dimensional arrays are organized as matrices which can be represented as the collection of rows and columns as array[M][N] where M is the number of rows and N is the number of columns.

To find the address of any element in a 2-Dimensional array there are the following two ways-

1. Row Major Order
2. Column Major Order

Row Major Order:

To find the address of the element using row-major order uses the following formula:

$$\text{Address of } A[I][J] = B + W * ((I - LR) * N + (J - LC))$$

I = Row Subset of an element whose address to be found,
J = Column Subset of an element whose address to be found,
B = Base address,
W = Storage size of one element store in an array(in byte),
LR = Lower Limit of row/start row index of the matrix(If not given assume it as zero),
LC = Lower Limit of column/start column index of the matrix(If not given assume it as zero),
N = Number of column given in the matrix.

Column Major Order:

To find the address of the element using column-major order use the following formula:

$$\text{Address of } A[I][J] = B + W * ((J - LC) * M + (I - LR))$$

I = Row Subset of an element whose address to be found,
J = Column Subset of an element whose address to be found,
B = Base address,
W = Storage size of one element store in any array(in byte),
LR = Lower Limit of row/start row index of matrix(If not given assume it as zero),
LC = Lower Limit of column/start column index of matrix(If not given assume it as zero),
M = Number of rows given in the matrix.

STRINGS:

- String is an array of characters.
- It is terminated by null character '\0'.
- It is usually stored as one-dimensional character array.
- '\0' character is stored to signify the end of the character array, after last character.

For ex:

"HELLO" is a string with 6 characters including the null character stored inside the computer memory.

The above string can be represented in memory as:

0	1	2	3	4	5
'H'	'E'	'L'	'L'	'O'	'\0'

Declaration of a String:

Strings are declared in C using the **char** data type. For ex:

char name[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 name[]={ 'H','E','L','L','O','\0'};

char s[]="HELLO";

The above declaration can be represented as:

name[0]	name[1]	name[2]	name[3]	name[4]	name[4]
'H'	'E'	'L'	'L'	'O'	'\0'

Q1. Can we declare size of string greater than no. of elements that are uninitialized?

Ans. YES.

Char str[10]="HELLO";

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]	str[9]
'H'	'E'	'L'	'L'	'O'	'\0'	'\0'	'\0'	'\0'	'\0'

Example: Program to demonstrate initialization of a string.

```
#include<stdio.h> int
main()
{
char name[5]={ 'L','I','O','N'};          //or char s[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]); return 0;
}
```

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**

Reading & Printing of Strings:

The strings can be accepted from the user using the following formatted functions:

1. Using scanf() and printf():

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> int  
main()  
{  
char str[15];  
printf("Enter a string: \n"); scanf("%s",str);  
printf("You typed the string:%s", str); return 0;  
}
```

Output 1:

Enter a string: HELLO
You typed the string: HELLO

Output 2:

Enter a string: HELLO WORLD You
typed the string: HELLO

Problem with scanf in reading a string and its solution:

Output 2 shows the drawback of using scanf(). 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 :

After modifying the scanf statement:

```
#include<stdio.h> int  
main()  
{  
char str[15];  
printf("Type a string: \n"); scanf("%[^\n]s",  
&str);  
printf("You typed the string:%s", str); return 0;  
}
```

Output:

Type a string: **Graphic Era Hill University**

You typed the string: **Graphic Era Hill University**

2. 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>`

```
int main()
{
char str[15];
printf("Type a string: \n");
gets(str);                                //same as scanf("%s", str);
printf("\nYou typed: ");
puts(str);                                //same as printf("%s", str);
return 0;
}
```

Output:

Type a string: **Programming in C**
You typed: **Programming in C**

3. Input/output String functions: getchar(), putchar()

getchar()--- repeatedly string is read by getchar() to read sequence of single characters unless terminating character is entered.
putchar()--- to display a characters on the screen.

Example: Program to illustrate the use of getchar() and putchar().

```
#include<stdio.h>
int main()
{
    int i=0,j,k=0;
    char str[20];
    char ch=getchar(); while(ch!='*')
    {
```

```

        str[i]=ch;
        i++;
        ch=getchar();
    }
    str[i]='\0';
    printf("The entered string is:\n"); while(str[k]!='\0')
    {
        putchar(str[k]); k++;
    }
    return 0;
}

```

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 string 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	strncmpi(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 excluding the null.

Example: Program to illustrate the use of strlen(). **#include<string.h>**

#include<stdio.h>

int main()

{

char str[20];

int len;

printf("Type a string:");

scanf("%[^\\n]s",str); len=strlen(str);

printf("\\nLength of the string %s is %d ", str,len); return 0;

```
}
```

Output:

Type a string: **HELLO**

Length of the string **HELLO** is **5** strcpy(s1,s2)

- This function copies the content of string s2 into another string s2.
- Putting text into a string:
strcpy(S, "This is String 1.");
- Copying a whole string from S to D:
strcpy(D, S);
- Copying the tail end of string S to D:
strcpy(D, &S[8]);

where D is destination and S is source.

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

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

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

```
int main()
```

```
{
```

```
    char s1[20],s2[20]; printf("Enter A  
    string: "); scanf("%^[^n]s",s2);  
    strcpy(s1,s2);          //Content of string s2 is copied into string s1  
    printf("Copied string:");  
    printf("%s",s1);  
    return 0;
```

```
}
```

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. where n is an integer

Assume that the following statement has been executed before each of the remaining code fragments.

- Putting text into the source string:
`strcpy(S, "This is String 1.");`
- Copying four characters from the beginning of S to D and placing a null at the end:
`strncpy(D, S, 4);`
`D[4] = '\0';`
- Copying two characters from the middle of string S to D:
`strncpy(D, &S[5], 2);`
`D[2] = '\0';`
- Copying the tail end of string S to D:
`strncpy(D, &S[8], 15);`
which produces the same result as `strcpy(D, &S[8]);`
- Example: Program to illustrate the use of `strncpy()`.

```
#include<string.h>
#include<stdio.h> int
main()
{
char s1[20],s2[20]; int
n;
printf("Enter a string: "); scanf("%[^\n]s",s2);
printf("\nHow many characters to be copied:");
scanf("%d",&n);
strncpy(s1,s2,n);           //Content of string s2 is copied into string s1
printf("Copied string: %s",s1);
return 0;
}
```

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. strcat() is used to concatenate a null-terminated string to end of another string variable.

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

```
<stdio.h>
#include <string.h>
int 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 return 0;
}
```

Output:

Enter the First String: **Graphic** Enter the
Second String: **Era** Concatenated String:
GraphicEra

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 strncat().

```
#include <stdio.h>
#include <string.h> int
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 return
0;
}

```

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

strcmp() is used to compare two strings. The strings are compared character by character starting at the characters pointed at by the two pointers. If the strings are identical, the integer value zero (0) is returned. As soon as a difference is found, the comparison is halted and if the ASCII value at the point of difference in the first string is less than that in the second (e.g. 'a' 0x61 vs. 'e' 0x65) a negative value is returned; otherwise, a positive value is returned. Examine the following examples.

```

char s1[25] = "pat";
char s2[25] = "pet";

```

diff will have a negative value after the following statement is executed. diff = strcmp(s1, s2);

diff will have a positive value after the following statement is executed. diff = strcmp(s2, s1);

diff will have a value of zero (0) after the execution of the following statement, which compares s1 with itself.

```
diff = strcmp(s1, s1);
```

Example-1: Program using strcmp() function.

```
#include <string.h>
#include<stdio.h> int
main()
{
char s1[30],s2[30]; printf("Enter the
first string: "); gets(s1);
printf("Enter the second string: "); gets(s2);
int diff= strcmp(s1,s2);
printf("%d",diff); return 0;
}
```

Output 1:

Enter the first string: **Program** Enter
the second string: **program 0**

Output 2:

Enter the first string: **Pat**
Enter the second string: Pet
-1

Example-2: Program using strcmp() function.

```
#include <string.h>
#include<stdio.h> int
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");
return 0;

}

```

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

Examine the following examples. char

s1[25] = "pat";

char s2[25] = "pet";

diff will have a negative value after the following statement is executed. diff = strncmp(s1, s2, 2);

diff will have a positive value after the following statement is executed. diff = strncmp(s2, s1, 3);

diff will have a value of zero (0) after the following statement. diff = strncmp(s1, s2, 1);

Example: Program using strncmp() function.

```

#include <string.h>
#include<stdio.h> int
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");
return 0;
}

```

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> int

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");
return 0;
}

```

Output:

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

strlwr(str):

This function is used to convert uppercase letters into lower case.

```

#include<stdio.h>
#include<string.h> int
main()
{
    char str[]="HeLLo";
    strlwr(str);
    printf("the string is: %s", str); return
    0;
}

```

OUTPUT: the string is: hello

strupr(str):

This function is used to convert lowercase letters into uppercase.

```

#include<stdio.h>
#include<string.h>
int main()
{
    char str[]="HeLLo";
    strupr(str);
    printf("the string is: %s", str);
}

```

```
    return 0;
```

```
}
```

OUTPUT: the string is: HELLO

strrev():

function is used to reverse the string.

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

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

```
main()
```

```
{
```

```
    char str[]="HELLO";
```

```
    strrev(str);
```

```
    printf("the string after reverse is: %s", str); return 0;
```

```
}
```

OUTPUT: the string is: OLLEH

Programs on strings without using inbuilt functions:

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

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

```
int 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); return
```

```
    0;
```

```
}
```

Output:

Type a string : **Blueberry**

Copied String is: **Blueberry**

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

```
#include<stdio.h>
int 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");
    return 0;
}
```

Output:

Enter first string: Graphic Enter
second string: Graphic Both the
strings are equal

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

```
#include<stdio.h>
int 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);
return 0;
}

```

Output:

Enter the First String: **Merry**
Enter the Second String: **goround**
Concatenated String is **Merry goround**

4. Develop a program to reverse a given string */

```

#include<stdio.h>
#include<string.h>
void main()
{
char str[100],temp;
int i=0,j=0;
printf("Enter the string 1");
gets(str);
j=strlen(str)-1;
while(i<j)
{
temp=str[i];
str[i]=str[j];
str[j]=temp;
i++;
j--;
}
printf("reverse of string is");
puts(str);
}

```

INPUT: hello

OUTPUT: olleh

5. Capitalize each alternate word in a string

```
#include<stdio.h>
int main()
{
    char s[100];
    int i,space=0;
    gets(s);
    for(i=0;s[i]!='\0';i++)
    {
        if(s[i]==32)
            space++;

        else if(space%2==0 && s[i]>='a'&&s[i]<='z')
        {
            s[i]=s[i]-32;
        }
    }
    printf("%s",s);
}
```

INPUT: graphic era hill university

OUTPUT: GRAPHIC era HILL university

6. Develop a program to check whether given string is palindrome or not.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str[100],str1[100],temp;
    int i=0,j=0;
    printf("Enter the string 1");
    gets(str);
    int n=strlen(str);
    while(str[i]!='\0')
    {
        str1[i]=str[i];
        i++;
    }
    str1[i]='\0';
    j=strlen(str)-1;
    i=0;
    while(i<j)
    {
        temp=str[j];
        str[j]=str[i];
        str[i]=temp;
        i++;
        j--;
    }
}
```

```

printf("reverse of string is ");
puts(str);
puts(str1);
int num=strcmp(str,str1);
printf("%d",num);
if (num)
{
    printf("string is not palindrome ");
}
else
printf("string is palindrome");
}

```

7. Develop a program to find the number of occurrences of each alphabet in a given string and display the same to the output screen. Assume that the string contains only alphabets.

```

#include <string.h>
int stringlength(char s[])
{
    int j;
    for(j=0;s[j];j++);

    return j;
}
void printfrequencyofcharacters(char s[])
{
    int i,j,count=0,n;

    n=stringlength(s);

    printf(" frequency count character in string:\n");

    for(i=0;i<n;i++)
    {
        count=1;
        if(s[i])
        {
            for(j=i+1;j<n;j++)
            {
                if(s[i]==s[j])
                {
                    count++;
                    s[j]='\0';
                }
            }

            printf(" '%c' = %d \n",s[i],count);
        }
    }
}

```

```

    }

int main()
{

    char s[1000];

    printf("Enter the string : ");
    gets(s);
    printfrequencyofcharacters(s);
    return 0;

}

```

POINTERS AND STRINGS:

Program 1:

```

#include<stdio.h> int
main()
{
char str[]="HELLO";
char str1[10];
//str=str1; //error
char *p=str; char
*q;
q=p;
while(*q!='\0')
{
printf("%p\t%c\n", (void *)q, *q); q++;
}
}

```

```
}
```

OUTPUT:

```
0x7ffe4c12674e    H
0x7ffe4c12674f    E
0x7ffe4c126750    L
0x7ffe4c126751    L
0x7ffe4c126752    O
```

// Demo Code copy text using pointers

```
#include<stdio.h> int
main()
{
    char name1[100] = "GEHU";      char name2[100];
    char *ptr1 ;
    char *ptr2 ;

    ptr1 = name1 ;
    ptr2 = name2 ;

    while (*ptr1 != '\0')
    {
        *ptr2 = *ptr1 ;
        ptr1++ ; ptr2++ ;
    }
    *ptr2 = '\0';
    printf("COPIED String is...\n%s", name2) ;
}
```

OUTPUT:

```
COPIED String is...
GEHU
```

PASSING STRING TO FUNCTIONS USING POINTERS:

Program 1: User Define Function to Copy a String:

```
#include<stdio.h> void
display(char*); int
main()
```

```

{
char str[10];
gets(str);
display(str); //can be &str, &str[0] return
0;
}
void display(char *p)
{
while(*p!="\0")
{
printf("%c",*p); p++;
}
}

```

Program 2: User Define Function to Compare two Strings

// Case Sensitive Version #include
<stdio.h>

```
int compare(char *, char *) ; int
```

```

main()
{
    char org[50] = "Arjun";
    char copy[50] ;

    printf("Enter the second String to Compare\n");
    scanf("%s", copy);

    if (compare(org,copy) == 0) printf("The two
    Strings are Equal \n"); else
        printf("The Two Strings are Un-equal \n"); return 0 ;
}

```

```

int compare(char *org, char *copy)
{
    int i = 0 ;

```

```

while ( (*org == *copy) && (*org != '\0' && *copy != '\0'))
{
    org++ ;
    copy++ ;
}

if (*org == '\0' && *copy == '\0') return
    0 ;
else
    return -1 ;
}

```

// Sample Output

Enter the second String to Compare Arjun
The two Strings are Equal

Enter the second String to Compare arjun
The Two Strings are Unequal

Two Dimensional Array of Characters:

```

#include <stdio.h> int
main()
{
    char months [12][25] =
    {"January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November",
    "December"} ;

    char *dow[] =
    {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };

    printf("Months of a Year\n") ; for
    (int i = 0; i < 12; i++) printf("%s\n",
    months[i]); printf("Days of the
    Week\n") ; for (int i = 0; i < 7; i++)
    printf("%s\n", dow[i]);
    return 0;
}

```

```
}
```

strtok():

The C library function `char *strtok(char *str, const char *delim)` breaks string `str` into a series of tokens using the delimiter `delim`.

- Following is the declaration for `strtok()` function:

`char *strtok(char *str, const char *delim)`

- Parameters:
`str` – The contents of this string are modified and broken into smaller strings (tokens).
`delim` – This is the C string containing the delimiters. These may vary from one call to another.
- Return Value:
This function returns a pointer to the first token found in the string. A null pointer is returned if there are no tokens left to retrieve.

The following example shows the usage of `strtok()` function: `#include`

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

int main () {
    char str[80] = "Graphic-Era-Hill-University"; const char
    s[2] = "-";
    char *token;

    /* get the first token */ token =
    strtok(str, s);

    /* walk through other tokens */
    while( token != NULL ) {
        printf( " %s\n", token );

        token = strtok(NULL, s);
    }
}
```



```
    return(0);  
}
```

OUTPUT:

Graphic
Era
Hill
University

atoi():

The C library function `int atoi(const char *str)` converts the string argument `str` to an integer (type `int`).

- `atoi()` converts a given string into its corresponding integer.
- This function returns that integer to the calling function.
- Therefore string should start with a number.
- This function will stop reading from the string as it gets a non-numerical character.

Syntax: `int atoi(const char*str);`

The following example shows the usage of `atoi()` function: `#include<stdio.h>`

`#include<stdlib.h> int`

`main()`

```
{  
    char str[10]="12345"; char  
    str1[10]="12345.57";  
    char str2[10]="12345a678"; int  
    i=atoi("12");  
    int a=atoi(str); int  
    b=atoi(str1); int  
    d=atoi(str2);  
    printf("%d\t",i);          //12  
    printf("%d\t",a);          //12345  
    printf("%d\t",b+5);        //12350  
    printf("%d\t",d);          //12345  
}
```