MODULE 3

Arrays

3.1 INTRODUCTION

Arrays: Array is a sequential collection of similar data items.

Pictorial representation of an array of 5 integers

10	20	30	40	50
A[0]	A[1]	A[2]	A[3]	A[4]

➢ An array is a collection of similar data items.

➢ All the elements of the array share a common name .

➢ Each element in the array can be accessed by the subsc ipt(or index)   nd   rray name.

➢ The arrays are classified as:

1. **Single dimensional array**

2. **Multidimensional array.**

3.1.1 Single Dimensional Array.

➢ A single dimensional array is a linear l st of related data items of same data type.

➢ In memory, all the data items are stored in contiguous memory locations.

Declaration of one-dimensional array(Single dimensional array)

Syntax:

> **datatype array_name[size];**

➢ **datatype** can be int,float,char,double.

➢ **array name** is the name of the array and it should be an valid identifier.

➢ **Size** is the total number of elements in array.
For example:

int a[5];

The above statement allocates 5*2=10 Bytes of memory for the array **a.**

a[0]   a[1]   a[2]   a[3]   a[4]

float b[5];

1

The above statement allocatests 5*4=20 Bytes of memory for the array **b.**

➤ Each element in the array is identified using integer number called as i**ndex.**

➤ If n is the size of array, the array index starts from **0** and ends at **n-1.**

Storing Values in Arrays

➤ Declaration of arrays only allocates memory space for array. But array elements are not initialized and hence values has to be stored.

➤ Therefore to store the values in array, there are 3 methods

    1. Initialization

    2. Assigning Values

    3. Input values from keyboard through **scanf()**

3.1.2 Initialization of one-dimensional array

➤ **Assigning the required values to an array elements   efo e p ocessing is called initialization.**

---

data type array_name[express on]={v1,v2,v3…,vn};

Where

   ✓ datatype can be char,int,float,double

   ✓ array name is the valid identifier

   ✓ size is the number of elements in array

   ✓ v1,v2,v3……..vn are values to be assigned.

➤ Arrays can be initialized at declaration time.
Example:

int a[5]={2,4,34,3,4};

2	4	34	3	4

   a[0]   a[1] a[2]   a[3] a[4]

➤ The various ways of initializing arrays are as follows:

    1. **Initializing all elements of array(Complete array initialization)**

    2. **Partial array initialization**

    3. Initialization without size

    4. String initialization

1. Initializing all elements of array:

➢ Arrays can be initialized at the time of declaration when their initial values are known in advance.

➢ In this type of array initialization, initialize all the elements of specified memory size.

➢ Example:

int a[5]={10,20,30,40,50};

10	20	30	40	50

2. Partial array initialization

➢ If the number of values to be initialized is less th n the size of rr y then it is called as partial array initialization**.**

➢ In such a case elements are initialized in the order from 0th element.

➢ The remaining elements w ll be n t al zed to **zero automatically by the compiler.**

➢ Example:
int a[5]={10,20};

10	20	0	0	0

3. Initialization without size

➢ In the declaration the array size will be set to the total number of initial values specified.

➢ The compiler will set the size based on the number of initial values.

➢ Example:

    int a[ ]={10,20,30,40,50};

➢ **In the above example the size of an array is set to 5**

4. String Initialization

➢ Sequence of characters enclosed within double quotes is called as string.

➢ The string always ends with NULL character(**\0**)

char s[5]="SVIT";	We can observe that string length is 4,but size is 5 because to store NULL character we need one more location.

So pictorial representation of an array **s** is as follows:

S	V	I	T	\0
S[0]	S[1]	S[2]	S[3]	S[4]

3.1.2 Assigning values to arrays

Using assignment operators, we can assign values to individual elements of arrays.
For example:

        int a[3];
                a[0]=10;
                a[1]=20;
                a[2]=30;

10	20	30

a[0]   a[1]   a[2]

Reading and writing single dimensional arrays.

To read array elements from keyboard we can use **scanf**() function as follows:

To read 0th element: scanf("%d",&a[0]);
To read 1st element: scanf("%d",&a[1]);
To read 2nd element: scanf("%d",&a[2]);
                                ……
                                …….
To read n^{th} element : scanf("%d",&a[n-1]);
In general
To read i^{th} element:
scanf("%d",&a[i]); where i=0; i<n; i++

To print array elements we can use **printf**() function as follows:

    o print 0th element: printf("%d",a[0]);
To print 1st element: printf("%d",a[1]);
To print 2nd element :printf("%d",a[2]);
            ……..
               ……..

To n^{th} element : printf("%d",&a[n-1]);
In general
To read i^{th} element:
printf("%d",a[i]); where i=0; i<n; i++

1.	Write a C program to read N elements from keyboard and to print N elements on screen.
|  | ```c
/* program to read N elements from keyboard and to print N elements
on screen */
#include<stdio.h>
void main()
{
        int i,n,a[10];
        printf("enter number of array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }

        Printf("array elements are\n"):
        for(i=0; i<n;i++)
        {
                prin tf("%d",a[i]);
        }
}
``` |
| 2. | Write a C program to find sum of n array elements . |
|  | ```c
/* program to find the sum of n array
elements.*/ #include<stdio.h>
void main()
{
        int i,n,a[10],sum=0;
        printf("enter number of array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n; i++)
        {
                scanf("%d",&a[i]);
        }

        for(i=0; i<n;i++)
        {
                sum=sum+ a[i];
``` |

```
        }
        printf("sum is %d\n",sum):

}
```

3.	Write a c program to find largest of n elements stored in an array a.

```c
#include<stdio.h>
void main()
{
        int i,n,a[10],big;
        printf("enter number of array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        big=a[0];
        for(i=0; i<n;i++)
        {
                if(a[i]>big)
                 big=a[i];
        }
        printf("the biggest element in an array is %d\n",big);
}
```

4.	**Write a C program to generate Fibonacci numbers using arrays.**

```c
#include<stdio.h>
void main()
{
        int i,n,a[10];
        a[0]=0; a[1]=1;
        printf("enter
        n\n");
        scanf("%d",&n);
        if(n==1)
        {
                printf("%d\t",a[0]);
        }
        if(n==2)
        {
                printf("%d\t %d\t",a[0],a[1]);
        }
        if(n>2)
```

```
        {
                printf("%d \t %d\t",a[0],a[1]);
                for(i=2;i<n;i++)
                {
                        a[i]=a[i-1]+a[i-2];
                        printf("%d\t",a[i]);
                }
        }
}
```

3.2 Searching

> The process of finding a particular item in the large amount of data is called searching.

> The element to be searched is called key element.

There are two methods of searching:

1] Linear search.

2] Binary search.

1] Linear Search:

> Linear search also called sequent al search is a simple searching technique.

> In this technique we search for a g ven key item in linear order i.e,one after the other from first element to last element.

> The search may be successful or unsuccessful.

> If key item is present, the search is successful, otherwise unsuccessful search.

1.	**Program to implement linear search.**
| | ```
#include<stdio.h>
void main()
{
        int i,n,a[10],key;
        clrscr( );
        printf("enter array elements\n");
        scanf("%d",&n); printf("enter
        array elements\n"); for(i=0;
        i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("enter the key
        element\n"); scanf("%d",,&key);

        for(i=0; i<n;i++)
        {if(key==a[i])
``` |

```
            {
                    printf("successful search\n");
                    exit(0);
            }
        }
    printf("unsuccessful search\n");
}
```

Advantages of linear search

➢ Very simple Approach.

➢ Works well for small arrays.

➢ Used to search when elements are not sorted.

Disadvantages of linear search

➢ Less efficient if array size is large

➢ If the elements are already sorted, linear search is not efficient.

2] Binary Search:

➢ Binary search is a simple and very efficient searching technique which can be applied if the items are arranged in either ascending or descending order.

➢ In binary search first element is considered as low and last element is considered as high.

➢ Position of middle element is found by tak ng f rst and last element is as follows.

$$mid=(low+high)/2$$

➢ Mid element is compared with key element, f they are same, the search is successful.

➢ Otherwise if key element is less than middle element then searching continues in left part of the array.

➢ If key element is greater than middle element then searching continues in right part of the array.

➢ The procedure is repeated till key item is found or key item is not found.

---

write a C program to perform binary search on the array of integers

```
/* C program to search a name in a list of names using Binary
searching technique*/
    #include<stdio.h>
    void main()
    {
            int i, n, low, high, mid,a[50],key;
            printf("enter the number of elements\n"):
            scanf("%d",&n);
            printf("enter the elements\n");
            for(i=0;i<n;i++)
            {
                    Scanf("%d",&a[i]);
            }

            printf("enter the key element to be
            searched\n"); scanf("%d",&key);
```

```
                    low=0;
                    high=n-1;
                    while(low<=high)
                    {
                            mid=(low+high)/2;
                            if(key==a[mid])
                            {
                                    printf("successful search\n");
                                    exit(0);
                            }
                            if(key<a[mid])
                            {
                                    high=mid-1;
                            }
                            else
                            {
                                    low=mid+1;
                            }
                    }
                    printf("unsuccesfull seasrch\n");
        }
```

Advantages of binary search
1. Simple technique
2. Very efficient searching
   technique Disadvantages
1. The elements should be sorted.
2. It is necessary to obtain the middle element, which are stored in array. If the elements are stored in linked list, this method cannot be used.

3.3 Sorting
➢
    he process of arranging elements in either ascending order or descending order is called Sorting.

Bubble Sort
➢
   This is the simplest and easiest sorting technique.
➢
   In this technique two successive elements of an array such as a[j] and a[j+1] are compared.
➢
   If a[j]>=a[j+1] the they are exchanged, this process repeats till all elements of an array are arranged in ascending order.
➢
   After each pass the largest element in the array is sinks at the bottom and the smallest element in the array is bubble towards top. So this sorting technique is also called as sinking sort and bubble sort.

PROGAM FOR BUBBLE SORT

```c
#include<stdio.h>
void main()
{
        int a[20],n,.temp,i,j;
        printf("enter the number of elements\n"):
        scanf("%d",&n);
        printf("enter the unsorted array elements\n");
        for(i=0;i<n;i++)
                scanf("%d",&a[ ]);
        for(i=1;i<n;i++)
        {
                for(j=0;j<n -i;j++)
                {
                    If( a[ i ] > a[ i+1 ] )
                    {
                        temp=a[j];
                        a[j]=a[j+1];
                        a[j+1]=temp;
                    }
                }
        }
            printf("the sorted elements arer\n");
        for(i=0;i<n;i++)
        printf(" %d\t", a[i]);
}
```

Write a C program to evaluate the polynomial using Horners method.

```
#include<stdio.h>
void main()
{
        int i,x,n,a[10],sum;
        printf("enter n:\n");
        scanf("%d",&n);
        printf("enter n+1 co efficient\n");
        for(i=0; i<=n;i++)
        {
                scanf("%d",&a[i]);
        }
        sum=a[n]* x;
        for(i=n-1; i>0; i--)
        {
                sum=(sum+a[i]) *x;
        }
        sum=sum+a[0];
        printf("sum of polynomial equation is %d",sum);
}
```

3.4 Two Dimensional arrays:

- In two dimensional arrays, elements will be arranged in rows and columns.

- To identify two dimensional arrays we will use two indices(say i and j) where I index indicates row number and j index indicates column number.

3.4.1Declaration of two dimensional array:

data_type array_name[exp1][exp2];

Or

data_type array_name[row_size][column_size];

- **data_type** can be int,float,char,double.
- **array name** is the name of the array.
- **exp1 and exp2** indicates number of rows and columns

For example:
        int   a[2][3];

✓ The above statements allocates memory for 3*4=12 elements i.e 12*2=24 bytes.

3.4.2 Initialization of two dimensional array

Assigning or providing the required values to a variable before processing is called initialization.

$$\text{Data\_type array\_name[exp1][exp2]=\{}$$

$$\{a1,a2,\ldots an\},$$
$$\{b1,b2,\quad .bn\},$$
$$\ldots\ldots\ldots$$
$$\ldots\ldots\ldots$$
$$\{z1,z2,\ldots zn\}$$
$$\};$$

➢ Data type can be int,float etc.

➢ exp1 and exp2 are enclosed within square brackets .

➢ both exp1 and exp2 can be integer constants or constant integer expressions(number of rows and number of columns).

➢ a1 to an are the values assigned to 1st row ,

➢ b1 to bn are the values assigned to 2nd row and so on.

Example:

int a[3][3]={

{10,20,30},

{40,50,60},

{70,80,90}

};

10	20	30
40	50	60
70	80	90

<u>Partial Array Initialization</u>

➢ If the number of values to be initialized is less than the size of array, then the elements are initialized from left to right one after the other.

➢ he remaining locations initialized to zero automatically.

➢ Example:

int a[3][3]={

{10,20},

{40,50},

{70,80}

};

10	20	0
40	50	0

70	80	0

1.	Write a c program to read & print 2d array as a Array.

```c
#include<stdio.h>
void main()
{
        int m,n,i,j,a[3][3];
        printf("enter number of rows and columns\n");
        scanf("%d %d",&m,&n); printf("eneter array
        elements\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("array elements are\n");
        for(i=0;i<m; ++)
        {
                for(j=0;j<n;j++)
                {
                        printf("%d",a[i][j]);
                }
                printf("\n");
        }
}
```

2	Write a c program to add two matrices.

```c
#include<stdio.h>
void main()
{
        int m,n,i,j,a[3][3],b[3][3] ,c[3][3];
        printf("enter number of rows and columns\n");
        scanf("%d %d",&m,&n); printf("enter array a
        elements\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
```

```
                                        printf("enter array b elements\n");
                                        for(i=0;i<m;i++)
                                        {
                                                for(j=0;j<n;j++)
                                                {
                                                        scanf("%d",&b[i][j]);
                                                }
                                        }

                                        for(i=0;i<m;i++)
                                        {
                                                for(j=0;j<n;j++)
                                                {
                                                        c[i][j]=a[i][j]+b[i][j];
                                                }
                                        }
                                        printf("resultant matrix c is \n");

                                        for(i=0;i<m;i++)
                                        {
                                                for(j=0;j<n;j++)
                                                {
                                                        pr ntf("%d\t",c[i][j]);
                                                }
                                                printf("\n");
                                        }
                                }
```

3	Write a c program to copy one 2d array in to another 2d array

```
                #include<stdio.h>
                #include<conio.h>
                void main()
                {
                        int m,n,i,j,a[3][3],b[3][3];
                        clrscr();
                        printf("enter number of rows and columns\n");
                        scanf("%d %d",&m,&n); printf("enter array a
                        elements\n");
                        for(i=0; i<m; i++)
                        {
                                for(j=0; j<n; j++)
                                {
                                        scanf("%d", &a[i][j]);
                                }
                        }

                        for(i=0;i<m;i++)
                        {
                                for(j=0;j<n;j++)
```

```
                {
                        b[i][j]=a[i][j];
                }
        }
        printf("matrix b is \n");

        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        printf("%d\t",b[i][j]);
                }
                printf("\n");
        }
}
```

4	Write a c program to find biggest element in a m t ix or 2D rr y.

```
#include<stdio.h>
void main()
{

        int m,n,i,j,a[3][3];
        clrscr();
        printf("enter number of rows and columns\n");
        scanf("%d %d",&m,&n); printf("enter array
        elements\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        big=a[0][0];
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        if(big>a[i][j])
                        big=a[i][j];
                }
        }
        printf("big is %",big);
}
```

5	**Write a C program to implement Matrix Multiplication**

```c
#include<stdio.h>
void main()
{
        int m,n,i,j,sum,p,q,k,a[3][3],b[3][3],c[3][3];
        printf("enter number of rows and columns of matrix a \n");
        scanf("%d %d",&m,&n);
        printf("enter number of rows and columns of matrix b \n");
        scanf("%d %d",&p,&q);
        if(n!=p)
        {
                printf("multiplication not possible\n"):
                exit(0);
        }

        printf("enter matrix   a elements\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }

        printf("enter array b elements\n");
        for(i=0;i<p;i++)
        {
                for(j=0;j<q;j++)
                {
                        scanf("%d",&b[i][j]);
                }
        }

        for(i=0;i<m;i++)
        {
                for(j=0;j<q;j++)
                {
                        {
                                c[i][j]=0;
                                for(k=0;k<n;k++)
                                {
                                        c[i][j]= c[i][j]+a[i][k]*b[k][j];
                                }

                        }
                }
        }
        printf("resultant matrix a is \n");

        for(i=0;i<m;i++)
```

```c
        {
                for(j=0;j<n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
        printf("resultant matrix a is \n");

        for(i=0;i<p;i++)
        {
                for(j=0;j<q;j++)
                {
                        printf("%d\t",b[i][j]);
                }
                printf("\n");
        }
printf("resultant matrix a is \n");

        for(i=0;i<m;i++)
        {
                for(j=0;j<q;j++)
                {
                        pr ntf("%d\t",c[i][j]);
                }
                printf("\n");
        }

}
```

6	Write a program to find sum of each row and sum of each column
|   | ```c
#include<stdio.h>
void main()
{

        int m,n,i,j,rsum,csum,a[3][3];
        printf("enter number of rows and columns\n");
        scanf("%d %d",&m,&n); printf("enter array
        elements\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        for(i=0;i<m;i++)
        {
                rsum=0;
``` |

```
                            for(j=0;j<n;j++)
                            {
                                    rsum=rsum+a[i][j];
                            }
                            printf("sum is %d",rsum);
                    }

                    for(i=0;i<m;i++)
                    {
                            csum=0;
                            for(j=0;j<n;j++)
                            {
                                    csum=csum+a[i][j];
                            }
                            printf("sum is %d",csum);
                    }

            }
```

7	Write a C program to add all 2D array elements

```
        #include<stdio.h>
        void main()
        {
                int m,n,i,j,sum=0,a[3][3];
                printf("enter number of rows and columns\n");
                scanf("%d %d",&m,&n); printf("enter array
                elements\n");
                for(i=0;i<m;i++)
                {
                        for(j=0;j<n;j++)
                        {
                                scanf("%d",&a[i][j]);
                        }
                }
                for(i=0;i<m;i++)
                {

                        sum=sum+a[i][j];

                }
                printf("sum is %d",rsum);
        }
```

FUNCTIONS

➢ A function as series of instructions or group of statements with one specific purpose.

➢ A function is a program segment that carries out some specific, well defined task.

➢ A function is a self contained block of code that performs a particular task.

4.1 Types of functions

➢ C functions can be classified into **two** types,

1. Library functions /pre defined functions /standard functions /built in functions
2. User defined functions

1. Library functions /pre defined functions /stand rd functions/Built in Functions

➢ These functions are defined in the **library of C compiler** which re used frequently in the C program.

➢ These functions are written by designers of c compiler.

➢ C supports many built in functions like
- Mathematical functions
- String manipulation functions
- Input and output functions
- Memory management functions
- Error handling functions

➢ EXAMPLE:

- pow(x,y)-computes x^y
- sqrt(x)-computes square root of x
- printf()- used to print the data on the screen
- scanf()-used to read the data from keyboard.

2. User Defined Functions

➢ The functions written by the programmer /user to do the specific tasks are called user defined function(UDF's).

➢ The user can construct their own functions to perform some specific task. This type of functions created by the user is termed as User defined functions.

4.2 Elements of User Defined Function

The Three Elements of User Defined function structure consists of :

1. **Function Definition**
2. **Function Declaration**
3. **Function call**

1. Function Definition:

➢ A program Module written to achieve a specific task is called as function definition.

➢ Each function definition consists of two parts:

   i.   Function header

  ii.   Function body

General syntax of function definition

Function Definition Syntax	Function Definition Example

datatypefunctionname(parameters)
 	
{
 	
    declaration part;
 	
    executable part;
 	
    return statement;
 	
}
	void add()
	
	{
	
	    int sum,a,b;
	
	    printf(“enter a and b\n”);
	
	    scanf(“%d%d”,&a,&b);
	
	    sum=a+b;
	
	    printf(“sum is %d”,sum);
	
	}

i.   Function header

   Syntax

 datatype functionname(parameters)

➢ It consists of **three** parts

   a) Datatype:

      ✓ **The data type can be int,float,char,double,void.**

      ✓ This is the data type of the value that the function is expected to return to calling function.

   b) <u>functionname</u>:

      ✓ The name of the function.

        ✓ It should be a valid identifier.

c) parameters

        ✓ The parameters are list of variables enclosed within parenthesis.

        ✓ The list of variables should be separated by comma.

        Ex: **void add(int a, int b)**

➢ In the above example the return type of the function is **void**

➢ the name of the function is **add** and

➢ The parameters are **'a' and 'b'** of type integer.

ii. Function body

➢ The function body consists of the set of instructions enclosed between { **and** } .

➢ The function body consists of following th ee elements:

    a) **declaration part:** variables used in function body.

    b) **executable part:** set of Statements or inst uctions to do specific activity.

    c) **return :** It is a keyword,it is used to **return control back to calling**

         function. If a function is not return ng value then statement is:

                 return;

        If a function is returning value then statement is:

                 return value;

2. Function Declaration

➢ The process of declaring the function before they are used is called as function declaration or function prototype.

➢ function declaration Consists of the data type of function, name of the function and parameter list ending with semicolon.

Function Declaration Syntax
datatypefunctionname(type p1,type p2,.........type pn);
Example
int   add(int a, int b);
void   add(int a, int b);

Note: The function declaration should end with a <u>semicolon ;</u>

3. Function Call:

➤ The method of calling a function to achieve a specific task is called as function call.

➤ A function call is defined as **function name followed by semicolon.**

➤ A function call is nothing but invoking a function at the required place in the program to achieve a specific task.

Ex:

void main()

{

      add( ); // function call without parameter

}

4.3 Formal Parameters and Actual Paramete s

- **Formal Parameters:**

➤ The variables defined in the **funct on header of function definition** are called **formal** parameters.

➤ All the variables should be separately declared and each declaration must be separated by **commas.**

➤ The formal parameters **receive the data from actual parameters.**

- **Actual Parameters:**

➤ The variables that are used when a function is invoked)in function call) are called **actual** parameters.

➤ Using actual parameters, the data can be transferred from calling function.

to the called function.

➤ he corresponding **formal** parameters in the **function definition** receive them.

➤ The **actual** parameters and **formal** parameters must match in number and type of data.

- **Differences between Actual and Formal Parameters**

Actual Parameters	Formal Parameters
Actual parameters are also called as **argument list.** Ex: add(m,n)	Formal parameters are also called as **dummy parameters. Ex:**int add(int a, int b)
The variables used in function call are called as actual parameters	The variables defined in function header arecalled fo mal parameters
Actual parameters are used in calling function when a function is called or invoked Ex: add(m,n) Here, m and n are called actual parameters	Formal p r meters re used in the function he der of a called function. Example: int add(int a, int b) { ……….. } Here, a and b are called formal parameters.
Actual parameters sends data to the formal parameters Example:	Formal parameters receive data from the actual parameters.

4.4   Categories of the functions

1. **Function with no parameters and no return values**
2. **Function with no parameters and return values.**
3. **Function with parameters and no return values**
4. **Function with parameters and return values**

1. Function with no parameters and no return values

1. Function with no parameters and no return values **(void function without parameter)**	
Calling function	**Called function**

/*program to find sum of two numbers
using function*/
#include<stdio.h>
void add( );
void main( )
{
     add( );
}
	void add ( )
	{
	    int sum;
	    printf("enter a and b values\n");
	    sc nf("%d%d",& ,&b);
	    sum= +b;
	    p intf("\n The sum is %d", sum);
	    etu n;
	}

- ✓ In this category **no data** is transferred from **calling function to called function**, hence called function cannot receive any values.
- ✓ In the above example,no arguments are passed to user defined function **add( ).**
- ✓ Hence no parameter are defined in function header.
- ✓ When the control is transferred from calling function to called function a ,and b values are read,they are added,the result is printed on monitor.
- ✓ When return statement is executed ,control is transferred from called function/add to calling function/main.

2. **Function with parameters and no return values (void function with parameter)**	
Calling function	**Called function**
```/*program to find sum of two numbers using function*/ #include<stdio.h> void add(int m, int n); void main() { int m,n; printf("enter values for m and n:"); scanf("%d %d",&m,&n); add(m,n); } ```	```void add(int a, int b) { int sum; sum = +b; printf("sum is:%d",sum); eturn; } ```

✓ In this category, **there is data transfer** from the calling function to the called function using parameters.

✓ **But there is no data transfer from** called function to the calling function.

✓ The values of actual parameters m and n are copied into formal parameters a and b.

✓ The value of a and b are added and result stored in sum is displayed on the screen in called function itself.

3. Function with no parameters and with return values	
Calling function	**Called function**

/*program to find sum of two numbers using function*/
 	
    #include<stdio.h>
    int add();
    void main()
    {
        int result;
        result=add( );
        printf("sum is:%d",result);
    }
	int add( ) /* function header */
	
	{
	
	        int a,b,sum;
	        printf("enter values for a and b:");
	        sc nf("%d %d",&a,&b);
	        sum= +b;
	         etu n sum;
	
	}

- ✓ In this category **there is no data transfer from the calling function to the called function.**
- ✓ But, there is data transfer from called function to the calling function.
- ✓ No arguments are passed to the function add( ). So, no parameters are defined in the function header
- ✓ When the function returns a value, the calling function receives one value from the called function and assigns to variable result.
- ✓ The result value is printed in calling function.

4. Function with parameters and with return values	
Calling function	**Called function**

/*program to find sum of two numbers using function*/
 	
#include<stdio.h>
int add();
void main()
{
    int result,m,n;
    printf("enter values for m and n:");
    scanf("%d %d",&m,&n);
    result=add(m,n);
    printf("sum is:%d",result);
}
	int add(int a, int b) /* function header */
	
	{
	
	   int sum;
	   sum= +b;
	   etu n sum;
	
	}

✓ In this category, there is data transfer between the calling function and called function.

✓ When Actual parameters values are passed, the formal parameters in called function can receive the values from the calling function.

✓ When the add function returns a value, the calling function receives a value from the called function.

✓ he values of actual parameters m and n are copied into formal parameters a and b.

✓ Sum is computed and returned back to calling function which is assigned to variable result.

4.5 Passing parameters to functions or Types of argument passing

The different ways of passing parameters to the function are:

✓ **Pass by value or Call by value**

✓ **Pass by address or Call by address**

1. Call by value:

➢ In call by value, the values of actual parameters are copied into formal parameters.

➢ The formal parameters contain only a copy of the actual parameters.

➢ So, even if the values of the formal parameters changes in the called function, the values of the actual parameters are not changed.

➢ The concept of call by value can be explained by considering the following program.

Example:

```c
#include<stdio.h>
void swap(int a,int b);
void main()
{
    int m,n;
    printf("enter values for a and b:");
    scanf("%d %d",&m,&n);
    printf("the values before swapping a e m=%d n=%d \n",m,n);
    swap(m,n);
    printf("the values after swapp ng are m=%d n=%d \n",m,n);
}


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

➢ Execution starts from function main( ) and we will read the values for variables m and n, assume we are reading 10 and 20 respectively.

➢ We will print the values before swapping it will print 10 and 20.

➢ The function swap( ) is called with actual parameters m=10 and n=20.

➢ In the function header of function swap( ), the formal parameters a and b receive the values 10 and 20.

➢ In the function swap( ), the values of a and b are exchanged.

➤ But, the values of actual parameters m and n in function main( ) have not been exchanged.

➤ The change is not reflected back to calling function.

2. Call by Address

➤ In Call by **Address,** when a function is called, the addresses of actual parameters are sent.

➤ In the called function, the formal parameters should be declared as pointers with the same type as the actual parameters.

➤ The addresses of actual parameters are copied into form l p   ameters.

➤ Using these addresses the values of the actu l p r meters c n be changed.

➤ This way of changing the actual pa amete s indi ectly using the addresses of actual parameters is known as pass by add ess.

Example:

#include<stdio.h>

void swap(int a,int b);

void main()

{

    int m,n;

    printf("enter values for a and b:");

    scanf("%d %d",&m,&n);

    printf("the values before swapping are m=%d n=%d \n",m,n);

    swap(&m,&n);

    printf("the values after swapping are m=%d n=%d \n",m,n);

  }

  void swap(int*a, int*b)

   {

    int temp;

    temp=*a;

    ***a=*b;**

    ***b=temp;**

   }

NOTE:

Pointer: A pointer is a variable that is used to store the address of another variable.

Syntax: datatype *variablename;

Example: int *p;

```
#include<stdio.h>
void main()
{
        inta ,*p;
        p=&a;
}
```

In the above program **p** is a **pointer** variable, which is storing the address of va iable **a**.

Differences between Call by Value and Call by reference

Call by Value	C ll by Address
When a function is called the **values of variables are passed**	When a function is called the **addresses of varia les are passed**
The type of formal parameters should be same as type of actual parameters	The type of formal parameters should be same as type of actual parameters, but they have to be declared as **pointers.**
Formal parameters contains the values of actual parameters	Formal parameters contain the addresses of actual parameters.

4.6 Scope and Life time of a variable

Scope of a variable is defined as the region or boundary of the program in which the variable is visible. There are two types

(i) Global Scope

(ii) Local Scope

i. Global Scope:

➢ The variables that are defined outside a block have global scope.

➢ That is any variable defined in global area of a program is visible from its definition until the end of the program.

➢ For Example, the variables declared before all the functions are visible everywhere in the program and they have global scope.

ii. Local Scope

   a. The variables that are defined inside a block have local scope.

   b. They exist only from thepoint of their declaration until the end of the block.

   c. They are not visible outside the block.

❖ **Life Span of a variable**

➢ The life span of a variable is defined as the period during which a variable is active during execution of a program.

For Example

❖ The life span of a global variable is the life span of the p og am.

❖ The life span of local variables is the life sp n of the function, they are created.

❖ **Storage Classes**

➢ *There are following storage classes which can be used in a C Program:*

    i. Global variables

    ii. Local variables

    iii. Static variables

    iv. Register variables

i. Global variables:

➢ These are the variables which are defined before all functions in global area of the program.

➢ Memory is allocated only once to these variables and initialized to zero.

➢ These variables can be accessed by any function and are alive and active throughout the program.

➢ Memory is deallocated when program execution is over.

ii. Local variables(automatic variables)

➢ hese are the variables which are defined within a functions.

➢ These variables are also called as automatic variables.

➢ The scope of these variables are limited only to the function in which they are declared and cannot be accessed outside the function.

iii. Static variables

➢ The variables that are declared using the keyword static are called static variables.

> The static variables can be declared outside the function and inside the function. They have the characteristics of both local and global variables.

> Static can also be defined within a function.
> *Ex:*

> *static int a,b;*

iv. *Register variables*

> Any variables declared with the qualifier register is called a register variable.

> This declaration instructs the compiler that the va iable under use is to be stored in one of the registers but not in main memo y.

> Register access is much faster compared to memo y ccess.
> Ex:

> register int a;

Recursion

> Recursion is a method of solv ng the pro lem where the solution to a problem depends on solutions to smaller nstances of the same problem.

> Recursive function is a function that calls itself during the execution.

> The two types of recursion are

1.Direct Recursion

2. Indirect Recursion

1.Direct recursion

> A recursive function that invokes itself is said to have direct recursion.

> For example factorial function calls itself hence it is called direct recursion.

2. Indirect recursion

> A function which contains call to another function which in turn contains calls another function, and so on.

Design of recursive function

> ***Any recursive function has two elements:***

        i.   **Base case**

        ii.   **General case**

i.   **Base case**
- The statement that solves the problem is called base case.
- Every recursive function must have at least one base case.
- It is a special case whose solution can be obtained without using recursion.
  A base case serves two purposes:

       i). It **act as a terminating condition**.

       ii). the recursive function obtains **the solution f om the base case it reaches**.

ii.   **General case:**
- The statement that reduces the size of the p oblem is c lled general case.
- This is done by calling the same function with educed size.
- In general recursive function of facto ial p oblem can be written as

5!=5*4!

      4!=4*3!

            3!=3*2!

              2!=2*1!

                  1!=1*0!

                     0!==1 This is a base case

                  1!=1*0!=1*1=1

             2!=2*1!=2*1=2

        3!=3*2!, 3!=3*2=6

     4!=4*3! , 4!=4*6=24

5!=5*4!, 5*=24,

120(answer)

n!=	1	if n=0	// Base case

n* (n-1)!   Otherwise // General case

Write (n-1)! as fact(n-1)

n!=   1   if n=0   // Base case
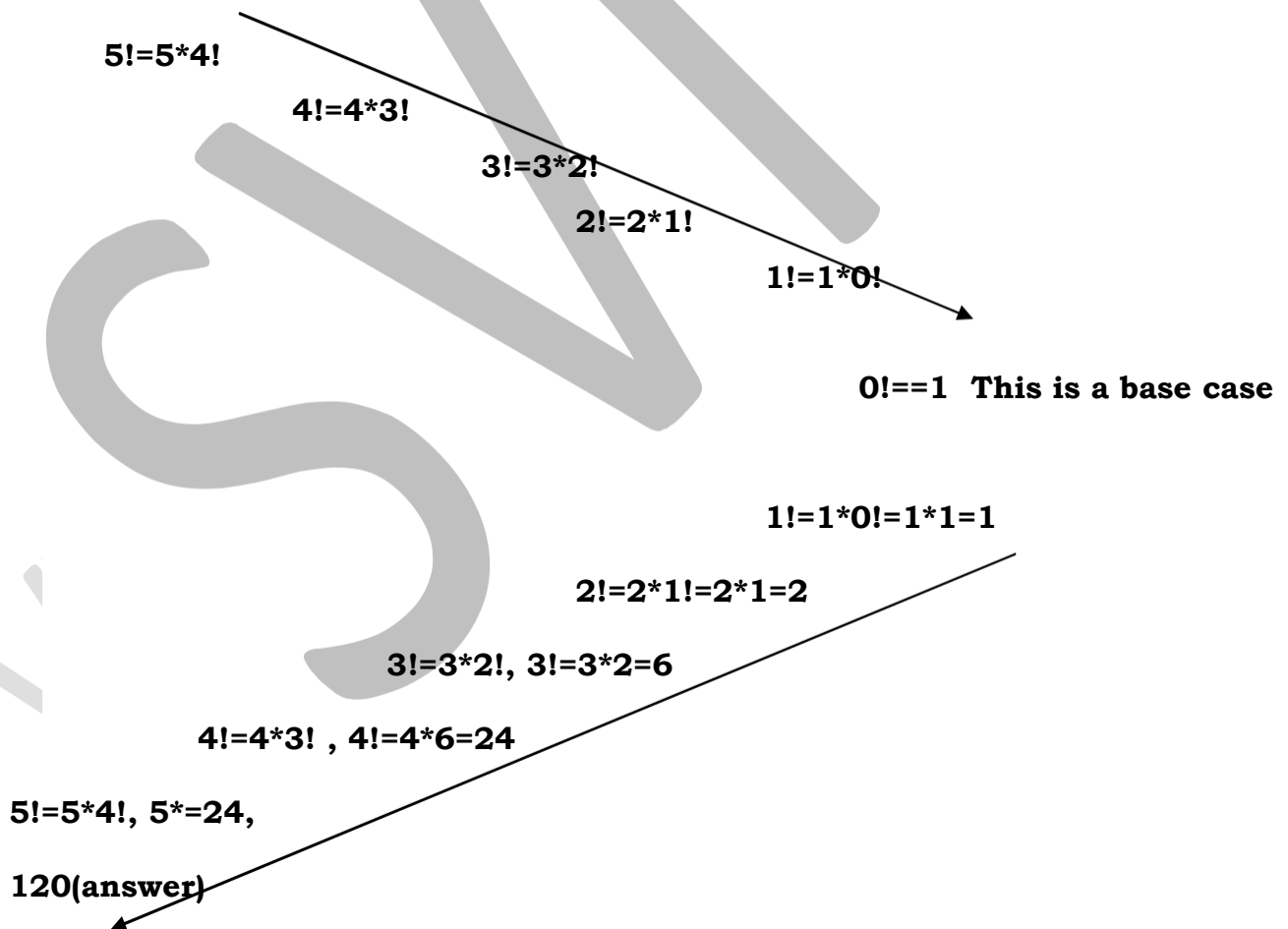
n* fact(n-1)   Otherwise // Gene al case
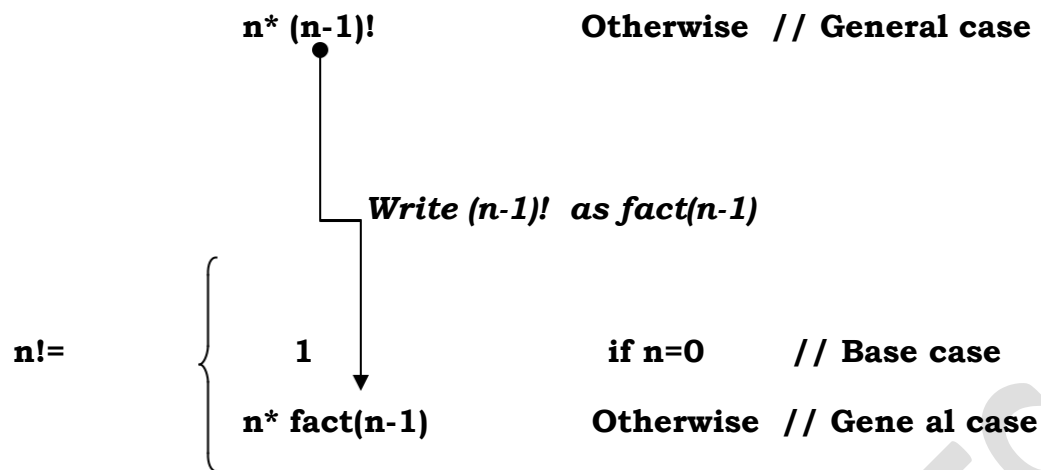
Example 1.

```
/******* Factorial of a given number using Recursion ******/
#include<stdio.h>
int fact(int n);
void main( )
{
    int num,result;
    printf("enter number:");
    scanf("%d",&num);
    result=fact(num);
    printf("The factorial of a number is: %d",result);
}
int fact(int n)
{
    if(n==0)
        return 1;
    else
        return (n*fact(n-1));
}
```

Module 5
Strings

Definition:

➢ A string is a sequence of characters within double quotes. A string const nt is lw ys termin ted y null character.

➢ A string is pictorially represented as follows:

a | S | V | I | T | \0
| 0   1   2   3   4
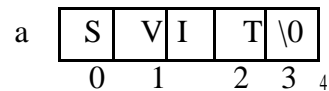
String Declaration:

➢ Like all other variable a string variable a string var able also has to e declared before it is used.

> **Syntax:** **char string_name[size];**

Example: char s[10];

The above declaration statement allocates 10 bytes of memory to string s as follows:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
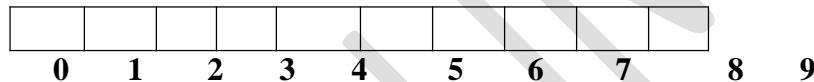
String Initialization:

Initialization is a process of assigning values to a string, before doing manipulation.
Strings are initialized in 4 ways:

1. Initializing character by character
2. Partial Array Initialization
3. Initialization without size
4. Initialization of Array with string

1.**Initializing character by character**	2. **Partial Array Initialization**

Consider following declaration and initialization
    char b[5]={ 'S','V','I','T'};
 	
The complier allocates 5 memory locations and these locations are initialized with the character in the order specified.
 	
b   S   V   I   T   \0
     0   1   2   3   4
	If the number of cha acte s to be initialized is less than the size of a ay then the emaining locations will be initialized to NULL as follows:
	Char b[5]={ 'H','I'};
	
	b H I \0 \0 \0
	     0 1 2 3 4

3.**Initialization without Size**	4.**Initialization of array with string**

If a string is declared without size then compiler will set the array size to the total number of initialized values.
char b[]={ 'S','V','I','T'};
 	
b   H   I   \0   \0   \0
     0   12   3   4
Size of b is 4
	char []="SVIT";
	
	In the above initialization the string length is 4 bytes but size is 5 bytes.
	
	b S V I T \0
	     0 1 2 3 4

String Input and Output functions:

➢ The strings can be read from the keyboard and can be displayed onto the monitor using carious functions: Input/Output Functions

Formatted Input Function **scanf()**	Formatted output Function **printf()**	UnFormattedInput Function **gets()**	UnFormatted Output Function **puts()**
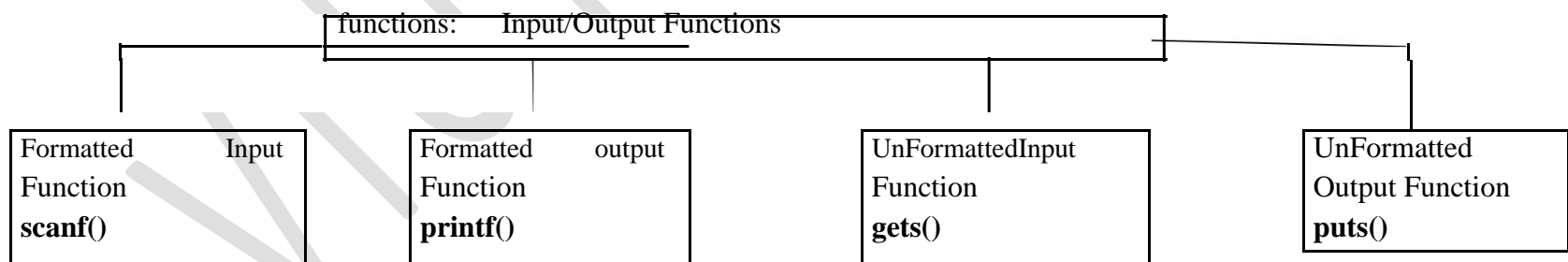
Formatted Input Function: scanf()	**Formatted Output Function: p intf()**
➤ The formatted input function is scanf().	➤ The formatted output function is printff().
➤ It reads a string from the keyboard	➤ It prints/displays a st ing which is stored on memory
➤ The format specifier is **%s**	locations on monitor
➤ **The string is terminated by NULL character(\0)**	➤ **Syntax: printf("%s",str);**
Syntax: scanf("%s",str);	➤ Example:
➤ Example:	char st [5]="SVIT";
char str[5]="SVIT";	p intf("%s",st);
scanf("%s",str);	

	S	V	I	T	\0
b					
	0	1	2	3	4

NOTE: scanf() cannot read spaces and any special symbols .e conversion code cannot read spaces, it will terminated as soon as space appear.

Write a program to read and print an string using scanf() and printf()

```
#include<stdio.h>
void main()
{
char str[20];
printf("enter the string\n");
scanf("%s",str);
printf("The entered string is \n");
printf("%s",str);
}
```

UnFormatted Input Function: scanf()	**UnFormatted output Function: puts()**
➢ The Unformatted input function is gets().	➢ The Unformatted output function is puts().
➢ It reads a sequence of characters(line) from the keyboard with spaces in between andstore them in memory locations.	➢ This function displa s all the character(line) stored in variable **str** on the monitor till it encounters **\0(Null Ch r cter)**
➢ **Syntax: gets(str);**	➢ **Synt x: gets(str);**

➢   Example:
char str[20];
printf(**"enter the string\n"**);
gets(str);
	➢ Ex mple:
	ch st [20]="HELLO";
	p intf(**"the st ing is \n"**);
	puts(st);

Write a program to read and print an string using gets() and puts()

```
#include<stdio.h>                         OutPut:
#include<string.h>
void main()                              Enter the string
{                                         HEL OHOWRU
char str[20];                            The entered string is
printf("enter the string\n");             HELLO HOW R U

gets(str);
printf("The entered string is \n");
puts(str);
}
```

H	E	L	L	O		H	O	W		R		U	\0				

Based on the kind of data processed, the I/O function are classified into
1. **Token Oriented I/O functions:**
2. **Line Oriented I/O functions**
3. **Character Oriented I/O functions**

1. **Token Oriented I/O functions:**

➢ The I/O functions processes individual units such as characte s,intege s,double v lues,float values and are separated by whitespaces characters. Since these individual units are called tokens,the functions th t perform these kind of operations are called *Token Oriented I/O* **functions.**

➢ The functions **scanf**() **and printf**() are *Token Oriented I/O* **functions.**

2. **Line Oriented I/O functions**

➢ The I/O functions that process entire line are called *l ne or ented I/O functions*.

➢ The functions **gets() and puts()** are *Line Oriented I/O* **funct ons.**

3. **Character Oriented I/O functions**

a) **getchar() and putchar()**

➢ To read a character from the keyboard and store this character into memory location, *getchar()* function is used.

➢ We have to press the ENTER KEY after typing character.
   Syntax: ch=getchar();

➢ To display a character stored in the memory on the screen ,*putchar()* function is used.
   Syntax:putchar(ch);

Write a C program to Find the section of student.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char sec;
  printf("Enter your section\n");
  scanf("%d",&sec);
```

OUTPUT

Enter Your Section
B

```
  sec=getchar();
 printf("Your section is \n");
  putchar(sec);
}
```

Your section is
B

  b) <u>getch(),getche(),putch()</u>

➤ The **getch()** function reads a character from the keyboard and copies it into specified memory location identified by **ch.**

➤ **Syntax: ch=getch()**

➤ The typed character will not be echoed(displayed) on the sc een if we use **getch()** function.

➤ No arguments are required for this function

❖ The **getche()** function reads a character from the keyboard and copies it into specified memory location identified by **ch.**

❖ **Syntax: ch=getche()**

❖ The typed character will be echoed(displayed) on the screen f we use **getche()** function. No arguments are required for this
     function

➤ The **putchar()** displays a character stored n memory location identified by variable **ch** on the screen.

➤ **Syntax: putch(ch)**

String handling Functions

SL. No	Name	Syntax	Example	Explanation
1	**strlen**	**int strlen (char str[ ]);**	**char str[15]="SVIT";** **int count;** **count=strlen(str);** | S	V	I	T	\0 | 0 1 2 3 4 **The example str va iable contains 4 characters S,V,I,T , hence count is 4**	-This function returns the length of the string str. -It counts all the characters until null character is encountered. ✓
2	**strcpy**	**strcpy(char dest[ ] , char src[ ]);**	**char src[5] ="SVIT";** **char dest[5];** **strcpy(dest ,src);** **src[0] src[1] src[2] src[3] src[4]** 	s	V	I	T	\0
S	V	I	T	\0	 **dest[0] dest [1] dest [2] dest [3] dest [4]**	**This function copies content from source string to destination string including \0.** ✔ **Size of dest string should be greater or equal to the size of source string src to store the entire source string.**

3.	strncpy	strcpy(char dest[ ] , char src[ ],int n);	char src[5] ="SVIT"; char dest[5]; strcpy(dest ,src,2); src[0] src[1] src[2] src[3] src[4] | S | V | I | T | \0 | | S | V | \0 | | | dest[0] dest [1] dest [2] dest [3] dest [4]	✔ This function copies n characters from source string to destination string . ✔ In thisexample only 2 characters are copied from src to dest.
4	strcat	strcat(char s1[ ] , char s2[ ]);	char s1[5]="SVIT"; char s2[5]="ECE"; strcat(s1,s2); | S | V | I | T | \0 | 0 1 2 3 4 | E | C | E | \0 | | 0 1 2 3 4 S1 | S | V | I | T | E | C | E | \0 | | | 0 1 2 3 4 4 6 7 8 9	✔ This function copies the all characters of s2 string to the end of s1 string. ✔ The delimiter of s1 is replaced by first character of s2. ✔ Size of s1 string should be greater or store the contents of both the string
5	strncat	strncat(char s1[ ] , char s2[ ],n);	char s1[5]="SVIT"; char s2[5]="ECE"; strncat(s1,s2,2); | S | V | I | T | \0 | 0 1 2 3 4	✔ This function copies the n characters of s2 string to the end of s1 string. ✔ The delimiter of s1 is replaced by first character of s2.

E	C	E	\0	
0	1	2	3	4

S1

S	V	I	T	E	C	\0			
0	1	2	3	4	4	6	7	8	9

In the above example only 2 cha acters(EC) from string s2 copies to st ing s1.

6	**strcmp**	**int strcmp(char s1[ ] , char s2[ ]);**	1) **Str ngs are equal** **S1[4]=”RAM”;** **S2[4]=”RAM”;** **Strcmp(S1,S2);**	**where:** **s1 is first string** **s2 is second string**

1) **Str ngs are equal**
S1[4]=”RAM”;
S2[4]=”RAM”;
Strcmp(S1,S2);

S1[0]	R	==	S2[0]	R
S1[1]	A	==	S2[1]	A
S1[2]	M	==	S3[2]	M
S1[3]	\0	==	S4[3]	\0

S1[0]==S2[0]
  R==R(ASCII value of R is compared)
similarly for other characters.
S1[3]==S2[3]
  \0==\0(ASCII value of \0 is compared and it is 0.)
2)String S1 is Lesser than String S2

where:
s1 is first string
s2 is second string
✔ **This function used to compare two strings.**
✔ **The comparison starts with first character of each string.**
✔ **This comparison continues till the corresponding character differ or until the end of the character is reached.**
✔ **The strcmp Returns 3values**
Possibly:
returns 0 if both strings are equal.
returns positive value ,if s1>s2
returns negative value if s1<s2

S1[4]="ABC";
S2[4]="BAC";
Strcmp(S1,S2);

S1[0]	A	==	S2[0]	B
S1[1]	B	==	S2[1]	A
S1[2]	C	==	S3[2]	C
S1[3]	\0	==	S4[3]	\0

S1[0]==S2[0]
   A==B(ASCII value of A is compared with ASCII value of B)
            .e 65==66 returns S1<S2

3)String S1 is Greater than String S2
S1[4]="BBC";
S2[4]="ABC";
Strcmp(S1,S2);

S1[0]	B	==	S2[0]	A
S1[1]	B	==	S2[1]	B
S1[2]	C	==	S3[2]	C
S1[3]	\0	==	S4[3]	\0

S1[0]==S2[0]
   A==B(ASCII value of A is compared with ASCII value of B)
            i.e 66==65 returns S1>S2

7	strncmp	int strcmp(char s1[ ] , char s2[ ], n);	1) **Strings are equal** S1[4]="RAM"; S2[4]="RAM"; strcmp(S1,S2,2);	where: s1 is first string s2 is second string

Detailed content for row 7:

int strcmp(char s1[ ] , char s2[ ], n);

1) **Strings are equal**
S1[4]="RAM";
S2[4]="RAM";
strcmp(S1,S2,2);

S1[0]	R	==	S2[0]	R
S1[1]	A	==	S2[1]	A
S1[2]	M	==	S3[2]	M
S1[3]	\0	==	S4[3]	\0

Only 2 characters from each   S1   and S2 is compared.
Other funct on s s milar to strcmp().

where:
s1 is first string
s2 is second string

✔ **This function used to comparen number of characters two strings.**
✔ **The comparison starts with first character of each string.**
✔ **This comparison continues till the corresponding character differ or until the end of the character is reached or specified number of characters have been tested..**
✔ **The strcmp Returns 3values**
Possibly:
returns 0 if both strings are equal.
returns positive value ,if s1>s2
returns negative value if s1<s2

| 8 | strrev( ) | void strrev(char str[ ]); |

Given string

S1	S	V	I	T	E	C	E	\0		
	0	1	2	3	4	5	6	7	8	9

strrev(s1
Reverse String

S1	E	C	E	T	I	V	S	\0		
	0	1	2	3	4	5	6	7	8	9

✔ **This function reverse all characters in the S1 except Null character.**
✔ **The original string is lost.**

			S1[6]==S1[0] S1[5]==S1[1] S1[4]==S1[2] S1[3]==S1[3] S1[2]==S1[4] S1[1]==S1[5] S1[0]==S1[6]	

Example Programs for string handling functions

strlen()

```
#include<stdio.h>
#include<string.h>
void main()
{ char name[15];
   int len;
  printf("Enter the string\n");
  gets(name);
   len=strlen(name);
    printf("\n The string length is %d",len);
}
```

OUTPUT
Enter the string
COMPUTER
The string length is 8

strrev()

```
#include<stdio.h>
#include<string.h>
void main()
{
       char str[]="INDIA";
       strrev(str);
       printf("string=%s",str);
}
```

OUTPUT
String=AIDNI

strcpy()	**strncpy()**

``
#include<stdio.h>
#include<string.h>
void main()
{
    char src[15],char dest[15];
   printf("Enter the source string\n");
   gets(src);
    strcpy(dest,src);
    printf("\n The copied string is \n");
    puts(dest);
}
``
	``
	#include<stdio.h>
	#include<st ing.h>
	void m in()
	{
	    ch s c[15],ch r dest[15];
	    int n;
	    p intf("Enter the source string\n");
	    gets(s c);
	    printf("Enter n");
	    scanf("%d",&n);
	     strncpy(dest,src);
	     printf("\n The copied string is \n");
	     puts(dest);
	}
	``

OUTPUT Enter the source string COMPUTER The copied string is COMPUTER	**OUTPUT** Enter the source string COMPUTER Enter n 3 The copied string is COM
strcat() ``` #include<stdio.h> #include<string.h> void main() ```	**strncat()** ``` #include<stdio.h> #include<string.h> void main() ```

<table>
<tr><td valign="top">

```
      {
          char S1[15],char S2[15];
        printf("Enter the string 1\n");
        gets(S1);
        printf("Enter the string 2\n");
        gets(S2);
         strcat(S1,S2);
         printf("\n The Concatenated string is \n");
        puts(S1);
      }
```

OUTPUT
Enter the string1
HELLO
Enter the string 2
ALL
The Concatenated string is
 HELLOALL

</td><td valign="top">

```
      {
          char S1[15],char S2[15];
        int n;
        printf("Enter the st ing 1\n");
        gets(S1);
        printf("Enter the string 2\n");
        gets(S2);
        p intf("Enter n");
        scanf("%d",&n);
         st ncat(S1,S2,n);
         printf("\n The Concatenated string is \n");
        puts(S1);
      }
```

OUTPUT
Enter the string1
HELLO
Enter the string 2
SVIT
Enter n
2
The Concatenated string is
 HELLOSV

</td></tr>
<tr><td valign="top">

strcmp()

```
      #include<stdio.h>
      #include<string.h>
      void main()
      {
          char S1[15],char S2[15];
```

</td><td valign="top">

strcnmp()

```
      #include<stdio.h>
      #include<string.h>
      void main()
      {
          char S1[15],char S2[15];
```

</td></tr>
</table>

int res;
printf("Enter the string 1\n");
gets(S1);
printf("Enter the string 2\n");
gets(S2);
res=strcmp(S1,S2);
if(res==0)
   printf("Strings are same\n");
else if(res>0)
   printf("String1 is greater than string2\n");
else
   printf("String1 is lesser than string2\n");
}
	int res,n;
	printf("Enter the st ing 1\n");
	gets(S1);
	printf("Enter the st ing 2\n");
	gets(S2);
	printf("Enter n");
	sc nf("%d",&n);
	es=st cmp(S1,S2,n);
	if(es==0)
	   p intf("Strings are same\n");
	else if(res>0)
	   printf("String1 is greater than string2\n");
	else
	   printf("String1 is lesser than string2\n");
	}

OUTPUT Enter the string1 HELLO Enter the string 2 HELLO The Strings are equal	**OUTPUT** Enter the string1 SVIT Enter the string 2 SVCE Enter n 2 String1 is greater than String2

Edit Set Conversion Code (%[…])

➢ Using edit set conversion code ,it is possible to enter a line of text with white spaces such as blank character,\t etc...

➢
  Syntax:scanf(%[.......] ,str)

                    edit characters

➤ edit characters represent the valid characters that are allowed in the string and should be enclosed within ' [' and ']'

Working:

➤ Each character read by *scanf* is compared with *edit character set*. ➤ If the character read is in **edit character set,** it is copied into **str**

➤ The above procedure is repeated as long as the character read is in **edit ch r cter set.**

➤ If the character read does not match with **edit character set**, the reading process is stopped nd remaining characters will still be available in keyboard buffer.

➤ If the first character read is not in **edit characters set**, the **scanf** is te min ted nd **'\0'** is copied into **str** indicating **null string.**

➤ The **edit characters set** is indicated by the symbol **caret(^)**

scanf("%[^\n]",str);   // read all characters except \n

Array of Strings

➤ Group of Strings is known as array of strings.

➤ To represent array of strings two dimensional array is required.

Declaration Syntax: **char string_name[row][column];**

Maximum length of each string

Number of Strings

Name of the Array

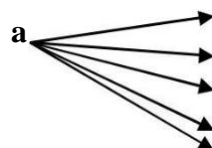Example: char a[5][20];

Where a is the string name

5 indicates at most 5 names can be stored

20 indicate that each name can have at most 20 characters

Initialization

char a[5][10]={

        "ARJUN",
        "BHARATHI",
        "SAMAYA",
        "SVIT",
        "VINAYAKA"
      };

➢
   Memory representation of the above initialization is as follows..

	0	1	2	3	4	5	6	7	8	9
0	A	R	J	U	N	\0				
1	B	H	A	R	A	T	H	I	\0	
2	S	A	M	A	Y	A	\0			
3	S	V	I	T	\0					
4	V	I	N	A	Y	A	K	A	\0	

Note: Example program is binary search