# Lecture 1

**Data Structure:**

A well-established logical & mathematical implementation of data is known as data structure. This structure is having an actual relationship with real world. This structure is quite simple. That one can effectively used & process. The Data when it may be required. Data Structure can be defined in two parts:-

1. Liner Data Structure
2. Non-Liner Data Structure
3. Liner Data Structure: The data structures in which accessing of data values are made in a sequential fashion are known as Liner Data Structure. Ex.- Array, Link List, Stack, Queue.
4. Non-Liner Data Structure: Here accessing of data element made in a non liner fashion or non continue fashion. Ex.- Tree, Graph

# Data Structure Operations

Liner performs various types of operations on the data. Some of these are as follows:-

1. Creation- Create any data structure.
2. Insertion- adding any data value in data structure.
3. Deletion- Delete any element from data structure.
4. Traversing & Visiting- read the data value of a data structure.
5. Sorting- Sort data structure in a particular fashion (Ascending & Descending).
6. Searching- Search any data value of a D.S.
7. Merging- Concatenate two D.S. in a single D.S.

# Data Structure Applications

1. Design & Controlling OS.
2. Design & Controlling the File Management.
3. Design & Controlling DBMS
4. Design & Controlling Process Management
5. Design & Controlling Memory Management
6. Design & Controlling Computer Language
7. Animation & Video Games
8. Any Computer Graphics Animation

# Stack

Stack is based in first in Last out. We can take the example of storing books in a box. Our first inserted book will be come out at last when all other inserted books have been popped. See the graphics: -Now you can see that first we have to bring out first two books ( 3 and 2 book ) then we can use book 1. This is the stack base. All function we will declare in the programs, they all will depend on this theory.

3 Book

2 Book

1 Book

# A simple Stack Program

#include<stdio.h>

#define max 3

int stack[max];

int top=-1;

void push(int a)

{

if(top==2)

cout<<"\n Stack is Full";

else

{ top++;

stack[top]=a;

}

}

void pop()

{ int x;

if(top==-1)

cout<<"\n Stack is Empty";

else

{ x=stack[top];

top--;

cout<< "\n deleted "<<x ;

}

}

void main()

{

push(1);

push(2);

push(3);

push(4);

pop();

pop();

pop();

pop();

}

**Infix , Postfix & Prefix**

### **Infix: -**

The notation which are we generally used is known as infix notation. In this type

of notation, we place the operator between operand.

Ex.- 1+2+4-3

### **Prefix: -**

Polish Notation. In this type of notation we type the operator before the operand.

Polish notation have serverl advantage over infix notation.

Ex.- +-\*abc

### **Postfix: -**

Reverse Polish Notation. In this type of notation we place the operator after the

operand.

Que.- Convert the following equation in prefix & postfix.

a+b\*c/d+e

1

Answer

a+1/d+e

2

a+2+e

3

3+e

#### Prefix

+3e

++a2e

++a/1de

++a/\*bcde

#### Postfix

3e+

a2+e+

a1d/+e+

abc\*d/+e+