Unit :- 01

16/08/2022

Data :- Raw facts and figures in the form of values are called data

Data Structure :- It is organising and arranging the data so that it becomes easily

OR

Logical & mathematical model of a particular organisation of data is called data structure.

⇒ Type of data structures

- Linear data structure
- Non-linear data structure
- Homogenous data structure
- Non-Homogenous data structure
- Primitive data structure
- Non primitive data structure
- static data structure
- Dynamic data structure

→ Linear data structure :- It is a type of a Data structure in which data items are arranged in linear sequence.
for ex :- array, stack, queue

→ Non linear data structure :- element are not in sequence for ex :- tree and graph

→ Homogeneous data structure :- elements are of same type ex :- array.

→ Non Homogeneous data structure :- elements may or may not be of same type ex :- structure

→ Primitive data structure :- It is provided by programming language as basic building blocks.
ex :- int, float, double, char

→ Non primitive data structure :- It is derived from primitive data structure ex :- stack, queue etc

→ Static data structure :- Memory is allocated at compile time

→ Dynamic data structures :- Memory is allocated at run time.

Stack example

```
class stack
{ int a[10], top;
  public:
  stack ();
  void push (int);
```

```
    int pop();
    int peek();
};
void stack :: stack()
{ top = -1; }
 void stack :: push (int x)
 {  top ++;
    a[top] = x;
 }
 int stack :: pop();
 {  int t;
    t = a[top];
    top --;
   return (t);
 }
 int stack :: peek ()
 { return a[top]; }
```

——— X ——— X ——— X ———

```
   class stack
    { int a[10], top,
      Public : stack();
       void push (int)
        int pop();
        void display ();
        int peek ();
    };
      stack :: stack()
     { top = -1; }
```

```
void stack :: push (int x)
  {
     a[top] = x;
    if ( top ≤ 9)
   {  top ++;
      a[top] = x;
   }
   else {
      cout << " space overflow";
      }
int stack :: pop()
{  if (top ≥ 0) {
     int t = a[top];
     top --;
     return t;
   }
  else { cout << " no elements left to pop"; }
}
```

⇒ Making dynamic allocation for array

```
  class stack
  { int *P,
    int size, top;
   public:
   stack () {}
   stack () {
      cout << " enter size of array"
      cin >> size
      p= new int [size]
  }
```

```
void push(int);
int pop();
int peek();
void disp();
~stack() {delete P;}
};
stack :: stack()
{ top=-1;}
void stack :: push(int x)
{ if (top < (size-1))
  { top++;
    a[top]=x;}
  else
  cout<<"stack is full"
}
```

```
int stack :: pop()
{ if (top<0)
   cout<<"stack overflow"
  else { int t = a[top];
   top--;
   return t;
  }
}
```

Converting Infix to Postfix

Operator maintains same order. Paranthesis not needed and priority of operator is not needed

Algorithm

1. Scan every character in infix string till end of the string
2. following steps are performed depending on the type of character scan
(i) if character scanned is a space then skip that character
(ii) if it is digit or alphabet then add it to target string
(iii) if it is opening paranthesis then add it to stack.
(iv) if it is an operator then compare the priority of topmost operator, topmost element from stack and the scanned character
3. following steps are acc. to precidence
(i) if the priority of operator is higher or same as scanned character/operator then add the operator to the target string and push scanned character to stack
(ii) if the operator is of low priority than scanned character then add it to stack
(iii) if it is closing parenthesis then operator in the stack are retrieve them and add it to target string till it doesn't read opening paranthesis
(iv) if source string exhaust then all operators in the stack are poped and add it to target string

Examples of Infix to Postfix

① a+b → ab+ [+]

② a+b÷c abc÷+ [÷ / +]

③ (a+b)/(c*d)

Sol^n → ab + cd-* /

| * |
| + (|
| (/ |

(4) ((A+B)*C − (D−E)) / (F+G)

Sol^n → ab + CDxDE − − FG+ /

| * |
| + 6 |
| (÷ |
| (* |

| + |
| * (|
| (* * |
| (/ |

26/08/2022

Infix to prefix.

Algorithm :-

1. Reverse the Infix string and start adding character to target string from last
2. Scan every character of source string following steps are performed depending upon the of character scanned
(i) If the character scanned is space then skip
(ii) If it is digit or alphabet then add to target string
(iii) If it is closing parenthesis then add to stack
(iv) If it is operator then compare the priority of topmost element from stack & scanned character, & acc. to the precedence perform following steps.
   (a) If operator has higher precedence then add operator to target string and push scanned character to stack
   (b) If operator has lower or same precedence then push scanned character to stack.
(v) If it is an opening paranthesis then operator from stack are poped and added to target string till closing paranthesis not reached.
3. If source string exhausted then add all operator from stack to target string

Example

(1) A+B
BA+
+AB

| + |

(2) A+B − C
⇒ CBA+−
⇒ − + AB /

| + |
| − |

(3) A+B*C
CB*A+
+A*B C //

| + |
| * |

(4) (A+B) * C(C−D)
DC − B*A+*
*+A*B − CD

| (|
| + |
| +) |
| *) * |

(5) A - B / (C * D + E)

⇒ E D C * + B / A -

⇒ - A / B + * C D E

Evaluate the postfix

⇒ Algorithm

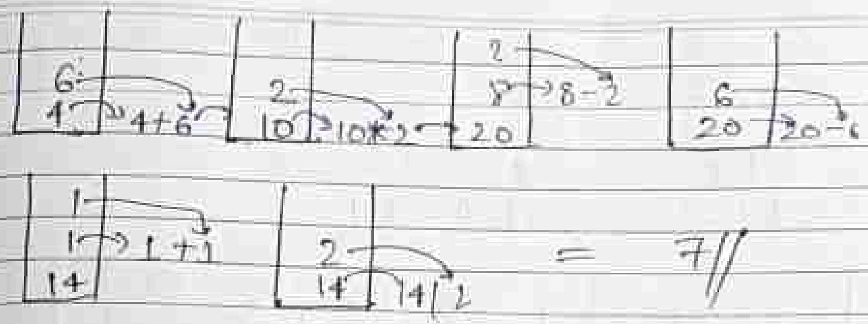
1. scan the postfix expression char by char
2. if char. scanned is an operator then push it to stack
3. if char. scanned is space then skip it
4. if it is operator then pop ~~tab~~ topmost elements from stack and operation is perform on operands

Example :

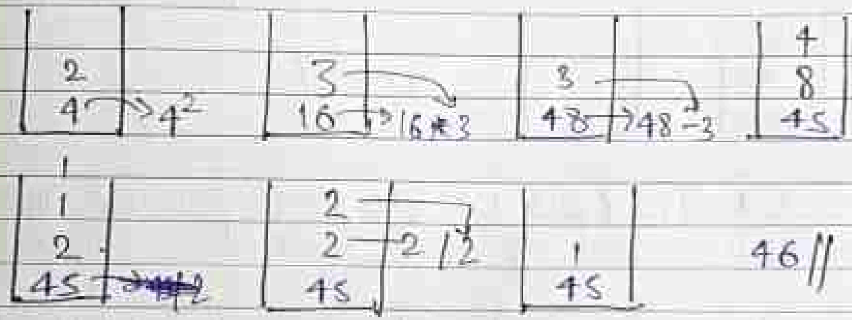
(1) ((A + B) * C - (D - E)) / (F + G)

Sol: Postfix :- AB+C*DE--FG+/

let A = 4, B = 6, C = 2, D = 8, E = 2, F = 1, G = 1



(2) 4 2 ^ 3 * 3 - 8 4 / 1 1 + / +



evaluation of prefix

(1) Reverse string

(2) jo 2nd element hoga usko pehle likhoge

g) ((A + B) * C - (D - E)) / (F + G)

G F + E D - C B A + * - /

/ - * + A B C - E D + F G

/ - * + A B C - E D + F G

A= 4, B=6, C=2, D=8, E=2,
F = 1, G=1

/-*+[illegible]ABC - [illegible]DE+FG
/-*+ 4 6 2 -8[illegible]2[illegible] +1 1

1 | 1+Y | 4 | 8 2 2 | 8-2 | 4 6 2 6 2 | 4+6 | 10 2 6 2 | 20 6 2

14	
2	7

7//

② (3+2)*(6/2) + (10*5) convert into prefix and postfix and then evaluate

③ evaluate + - * ^ 4 2 3 3 // 8 4 + 1 1 in prefix form

Sol^n 2 Postfix :-

32+62/*105*+

	)	)
)	/	*
+	(	(
(	*	+

evaluate :-

	2		5		
2	6	3	10	50	
3	5	5	15	15	65

=> 65//

Prefix :- 5 10 * 2 6 / 2 3 + * +

+ * + 3 2 / 6 2 * 10 5

		(
	(	*
(	*	*
*	*	*
*	*	

evaluation

		3			
	6	2	5		
10	2	3	3	15	
5	50	50	50	50	65

=> 65//

Sol^n 3 evaluating :-

			4				
			2	16			
	8		3	3	48		
1	4	2	3	3	3	45	
1	2	2	1	1	1	1	46

Queue

```
class queue
{ int q[10]
  int f, r;
  Public : queue ();
  void add_element (int);
  int del_element ();
};
queue : queue ()
  q[f] = -1; r = f = -1;
```

```
for (int i=0; i<10; i++)
    q[i] = -1;
}
void queue :: add_element (int a)
{ if (r<10)
  { if (r == -1)
      f = r = 0; q[r] = a;
    else r++;
      q[r] = a;
  }
  else
    cout << "queue is full";
}
int queue :: del_element ()
{  if (f>=0)
   {
     int t = q[f]
     q[f] = -1
     if (f == r) { f = r = -1; }
   else {
       f++; }
   return t;
   }
   else { cout << " queue is empty";
   return -1;
}
```

Dynamic allocation of array

```
class queue
{ int *q, size;
  int f, r;
  public: queue ();
   void add_element (int);
   int del_element ();
  ~stack ()
  { delete q; }
};
queue :: queue (int x = 10)
{ size = x;
   q = new int [size]
    r = f = -1
   for (int i=0; i<size; i++)
       q[i] = -1;
}
void queue :: add_element (int a)
{ if (r<size)
  { if (r == -1)
      f = r = 0; q[r] = a;
    else r++
     q[r] = a;
  }
  else
   cout << " queue is full ";
}
```

```
int queue :: del element ()
{ if (f>=0)
  {
    int t = q[f]
    q[f] = -1;
    if (f == r)
      f = r = -1;
    else {
      f++;
    }
    return t; }
  }
  else {
    cout << " queue empty";
    return -1;
  }
```

Circular Queue

```
Class queue
{ int q[10], f, r;
  Public : queue () { f = r = -1; }
  void add_ele (int);
  int del_ele ();
  void disp ();
};
void queue :: add_ele(int x)
{ if ((r == max-1 && f == 0) || f == r+1)
{ cout << " queue is full";
  return; }
  if (f == -1 && r == -1)
  {  f = r = 0 ; q[r] = x; }
  else if (r < 9)
  { r = r+1;
    q[r] = x; }
  else if (r == 9 && f != 0)
  { r = 0;
    q[r] = x;
  }
}

int queue :: del_ele()
{ int t = -1;
  if (f == -1) { cout << " queue is empty";
                 return t; }
  t = q[f]; q[f] = -1;
  if (f == r)
    f = r = -1;
  else if (f == 9)
    f = 0;
  else
  { f++;
  return t; }
}
```

D Queue

```
class dqueue
{ int q[10], f, r;
  public: dqueue();
  queue (-) {
  -- -        f=r=-1; }
   void add_elef(int);
   int del_eler();
};  dqueue::
 void ^add_ele_f (int x)
 {  if (f==0)
    { cout << " item can't be added from front";
    }
    if (f==-1)
       f=r=0;
    else
    { f--;
      q[f] = x;
    }
}void
int queue :: del_ele_r()
{ if (f==-1)
  { cout<<"queue is empty";
    return -1;
  }
   int v= q[r] ; q[r] = -1;
```

```
if (f==r)
    f=r=-1;
else
    r--;
return v;
}
```

Linked List

```
struct node
{ int d;
  node *next;
};
class linked_list
{ node *start;
  public: Linkedlist(){start = Null;}
  void add_ele(int);
  void del_ele(); void display();
};
void linkedlist :: add ele (int dd)
{   node *t = new node;
          t->d=dd;
          t->next = NULL;
        if (start == NULL)
            start=t;
        else { node * temp;
               temp = start;
```

only as sequential access in LL.

```
    while (temp->next != NULL)
     { temp = temp->next;}
       temp -> next = t;
}
```

```
void main()
{ clrscr();
  class linkedlist l;
   l.add_ele(50);
  getch();
};
```

```
void linkedlist :: del_ele (int dd)
{  if (start == NULL)
   { cout << "List is empty";
     return; }
     node *temp, *old;
     temp= start;
   if (start ->d==dd)
   {  start = start -> link/next;
       f=1;
     delete temp;
   }
else {
   while (temp != Null)
   { if (temp->d == dd)
     { old -> link/next = temp -> link/next;
       delete temp;
       f=1;
     }
     else { old = temp;
          temp = temp -> link/next;
        }
   }
}
if (f==0)
 { Cout << "element not found";}
}
```

```
void linked_list :: display ()
{ node * temp;
    start = temp;
    if (start = NULL)
      cout << "List is empty";
    else
      while (temp! = NULL)
          cout << temp->d;
          temp= temp -> next;
}
Void linked_list :: count ()
{ int c=0
   node * temp = start;
   while (temp != NULL)
   { c ++;
     temp = temp ->next
```

Adding node at begining

```
   struct node
  { int d;
    node * link ;
  };
   class linked-list
  { node * start
    Public : linked - list () {start = NULL;}
    void add_ begin (int) ; Void insert (int,i
```

```
void linked_ list :: add_ begin (int x)
{ node * temp = new node ;
    temp ->d = x;
    temp -> link = start;
    start = temp;
}
void linked list :: insert (int l, int x)
{ int c = count ();
  if ( l<= (c+1) && l>0 )
  { if ( l== 1 )
       add_begin (x);
    else if (l== (c+1))
       add ele (x);
    else {
       node * temp = start;
       node * t = new node;
        t -> d= x;
       for (int i=1, i<(l-1); i++)
       { temp= temp-> link;
       }
        t-> link = temp -> link;
        temp -> link = t;
     }
}
```

Stack using LL

```
struct node
{ int d;
  node *link;
}
class stack
{ node *tos;
  public:
    stack () { tos = NULL; }
    void push (int);
    int pop ();
};
void stack :: push (int x)
{ node *temp = new node;
    temp -> d = x;
    temp -> link = tos;
    tos = temp;
}
int stack :: pop ()  -->  if (tos == NULL)
{ node *temp = tos;       { cout << "Stack is
  int dd = temp -> d;              empty"; }
  tos = tos -> link;        return -1; }
  delete temp;
  return dd;
}
```

Queue using LL

```
Struct node
{ int d;
  node * link;
}
class queue
{ node *f, *r;
  public:
    queue () { f = r = NULL; }
    void add_node (int);
    int del_node ();
}
void queue :: add_node (int x)
{ node *temp = new node;
  temp -> d = x;
  temp -> link = NULL;
  if (r == NULL)
     r = f = temp;
  else
     r -> link = temp
       r = temp;
}
int queue :: del_node ()
{ if (f == NULL)
  { cout << "queue is empty";
    return -1; }
  else { node * temp = f;
```

S

```
    f = f -> link;
    int t = temp -> d;
    delete temp;
    if ( f == NULL)
        r = NULL;
    return t;
}
```

Input restricted DQueue

```
Struct node
{ int d ;
  node * link;
}

Class Dqueue
{ node *f, *r;
  Public:
    Dqueue () { f=r = NULL;}
  void add-node (int );
  int del-node ();
};

int del_node ()
{ if (r == NULL)
  { cout << " Queue is empty ";
    return -1;}
  int t;
  else if (f==r)
  { t = r->d;
    delete r;
    r = f = NULL;
    return t;
  }
  while (temp -> link -> link ! = NULL)
  { temp = temp -> link;
    r = temp;
    temp = temp -> link;
    delete temp;
    return t;
  }
```

16/09/22

Doubly linkedlist

```
Struct node
{ int d;
  node * prev, * next;
}
class dlinkedlist
{ node * start;
  public: dlinkedlist ()
          { start = Null;}
  void append-node (int );
  int delete (int );
};
```

```
# void dlinkedlist :: append_node (int x)
{ node *t, * temp;
  t= new node ;
  t ->d= x;
  if (start = =NULL )
  {  start = t;
     start ->prev = NULL;
  }
  else { temp = start;
    while (temp ->next != NULL )
    { temp = temp ->next; }
    temp => next= t;
    t -> prev = temp;
  }
}
void linkedlist :: delete (int x)
{   if (start = = NULL )
    { cout << "list is empty";
      return -1; }
  else { node *temp, *old,  temp = start;
        int  t,  f = 0;
        if (start ->d != x)
           start =NULL;  f = 1;
        else {
            start = start ->next;
            start ->prev= N ULL;
        }
  }
else{
  while (temp ->next != NULL)
  { old  =  temp;
    temp  = temp ->next;
    if ( temp ->d == x)
    {   f = 1;
      if (temp ->next == NULL )
        temp ->prev ->next = NULL;
      else {
          old ->next = temp -> next;
          temp ->nex -> prev =  old;
      }
    }
  }
}
if (f == 1) { t = temp -> d;
           delete temp;
           return t;
         }
else { cout << "node not found";
     return -1; }
}

 void dlinkedlist :: add_node_begin (int x)
 { node *temp = new node;
   temp ->d = x;
   if (start = = NULL)
   {   start = temp;
       start ->next = start ->prev = NULL;
   }
```

```
else { temp -> next = start;
#       start -> prev = temp;
        start = temp;
        start -> prev = NULL;
      }
}
void dlinklist :: insert (int l, int x)
{   int c = count ();
    if (l>0 && l< (c+2))
    {  if (l==1)
          add_node_beg (x);
       else if (l= (c+1))
          append (x);
       else { node *temp, *t;
          t = new node;
          t->d = x;
          temp = start;
          for (int i=1; i<(l-1); i++)
          { temp = temp->next; }
          t -> next = temp->next;
          t-> prev = temp;
          t-> next -> prev = t;
          temp ->next = t;
       }
    }
    else { cout << "Invalid location"; }
}
```

Searching

Linear Search

```
class array
{ int a[5], i
  public :
  void get_element (); void display ();
  void search element ();
}
  array ::
void get_element ()
{ int a, i;
  for( i=0; i<5; i++)
  { cout << "Enter the element at" << i << "index";
    cin >> a[i];
  }
}
void array :: display()
{ for (i=0; i<5; i++)
  { cout << "element at a" << i << "is" << a[i];
  }
}
void sarray :: search ()
{ int f=0, b;
  cout << "Enter value that you want to search";
  cin >> b;
```

```
for (i=0 ; i<5 ; i++)
}  if (b == a[i] )
   { cout << "element found ";
       f= 1;
     }
}
   if (f= =0) { cout << "element not found"; }
}
```

Binary search

```
class array
{ int a[5], i;
  public:
  void get_element();
  void display();
  void search();
}
void array :: get_element ()
{
    for (i=0; i<5; i++)
    { cout << "Enter element at "<< i << "index"
      cin >> a[i];
    }
}
void array :: display()
{     for (i=0; i<5; i++)
    { cout << "element at a" << i << " is " <<a[i]
    }
}
```

```
void array :: search ()
{  int l=0, u=4, m, f=0;

   for(m= (l+u)/2 ; l≤u; m= (l+u)/2 )

{   if (a[m] = =x)
    { cout << "element found";
      f=1 ;
      break;
    }
    else if (a[m]>x)
          u=m-1;
    else
         l=m+1;
}
  if (f= =0)
  { cout << "element not found"; }
}
```

Sorting

Selection Sort

```
 void sort()
{  int i, j;
   for (i=0 ; i<n; i++)
     } for (j= 1; j<n ;j++)
          } if (a[i]> a[j])
             { int c;
               c= a[i];
               a[i]= a[j];
               a[j] = c; }
```

Bubble sort

```
Void Bsort()
{ int temp =0;
for (int i=0; i<size-1; i++)
{
    for (int j=0; j<size-(i-1); j++)
    {
        if (a[j] > a[j+1])
        {   temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
        }
    }
}
}
```

Quick Sort : It's a recursive algorithm

Algorithm :

→ In first iteration

```
int j;
void qsort (int low, int high)
{  if (low < high)
   {  partition (low, high);
      qsort (low, j-1);
      qsort (j+1, high);
   }
}
```

```
Void partition (int lb, int ub)
{
    int pivot = a[lb];
    int down = lb+1;
    int up = ub;
    while (down <= up)
    { while (a[down] <= pivot)
      down ++;
      while (a[up] > pivot)
      up --;
      if (up > down)
      { int temp = a[up];
        a[up] = a[down];
        a[down] = temp;
      }
    }
    a[lb] = a[up]
    a[up] = pivot;
    j = up;
```

Merge Sort

A

4	5	6	10	14

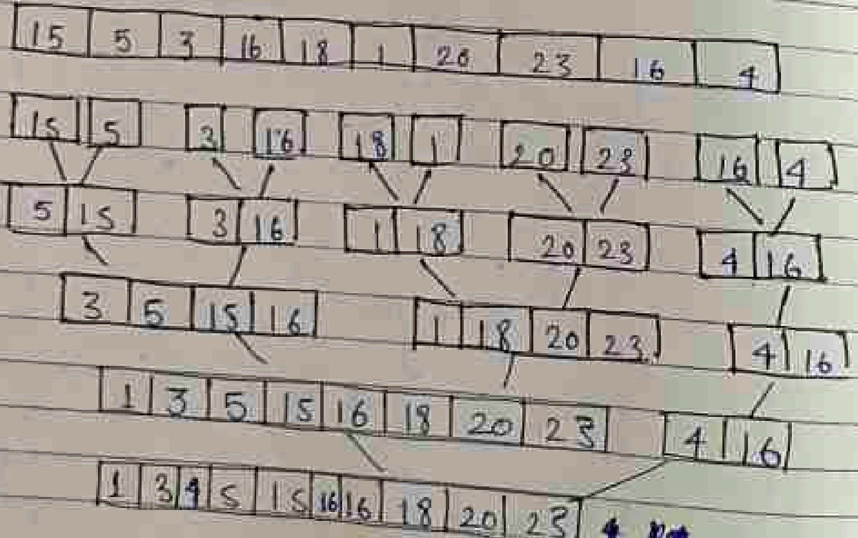
B

2	3	8	12	13	16	20

int i = j = k = 0;

```
m=5 , n=7;
int a[5], b[8], c[12];

while (i<m && j<n)
{  if (a[i] < b[j])
        c[k++] = a[i++];
   else
        c[k++] = b[j++];
}
while (i<m)
{     c[k++] = a[i++];
while (j<n)
      c[k++] = b[j++];
```




```
Void mergesort()
{
    int size = 1, i, j;
    int lb1, lb2, ub1, ub2, temp[n];
    While (size<n)
    { lb1 = 0;
      int x=0;
      While (lb1 + size <= n-1)
      { lb2 = lb1 + size;
        ub1 = lb2 - 1;

        if (lb2 + size-1 > n-1)
              ub2 = n-1;
        else
              ub2 = lb2 + size-1;
        i = lb1;
        j = lb2;

        While (i<=ub1 && j<=ub2)
        { if (arr[i] <= arr[j])
              temp[x++] = arr[i++];
          else
              temp[x++] = arr[j++];
        }
        While (i<= ub1)
              temp[x++] = arr[i++];
        lb1 = ub1+1 ; increment condition
```

Sorting b/w the elements

```
l= lb1 ;

while (k <=n-1)
    temp [k++] = arr [i++] ; //for single
                                 element

  for (x=0 ; x<=n-1 ; x++)
      arr [x) = temp [x]; // for copying

    size = size *2;
  }
}
```

11/10/2022

Insertion Sort

```
void array :: isort ()
{
  int i,j,k, temp;

   for (i=0 ; i<size; i++)
   {
      for (j=0; j<i ; j++)
      {
        if (a[j] > a[i])
        {
           temp =a[i] ;

   for (k=i ; k>j ; k--)
   {
      a [k] = a[k-1] ;
   } a[j]= temp ; }
   }
  }
 }
}
```

Radix Sort

The Radix sort algorithm works on by ordering each digit from least significant to most significant. In base 10, radix sort would sort by the digits in the one's place, then the ten's place and so on. To sort the value in each digit place, Radix sort employees counting sort as a subroutine

```
void Radix_sort ()
{
```

Tree

⇒ It is a non linear data structure
⇒ collection of nodes organised in hierarchical structure.

⇒ Node :- Each element of a tree is node.
⇒ Root :- Element that represent the base node of the tree
⇒ Leaf :- The node who does not have any child
⇒ Degree of node :- No. of node connected to a Node
⇒ Level :- root node ⇒ 0 level
Level of other node will be one more than root node
⇒ Depth or height of tree :- Maximum level of any leaf node in the tree is called depth or height.

⇒ Binary tree :- A finite set of element that are either empty/partitioned into three(3) disjoint subsets. The first subset contains a single element called the root of the tree. The other two subset are called left and right subtree of original tree

• A binary tree is called strictly binary tree if every non-leaf node is a binary tree has non-empty left and right subtree

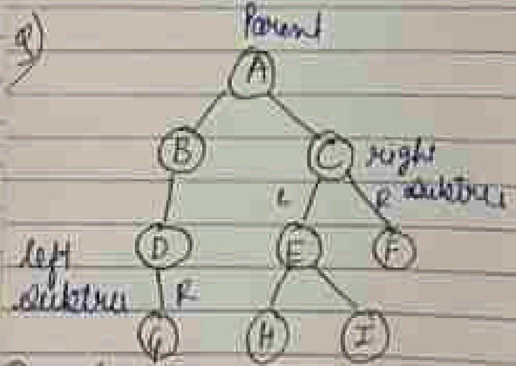
- A strictly binary tree all of whose leaf node are at same level is called complete binary tree

Traverse on tree

⇒ Pre order traversal :- visit the root node then traverse the left subtree than traverse the right subtree

General form :- PLR ⇒ ① Parent node
② Left subtree
③ Right subtree

Q)

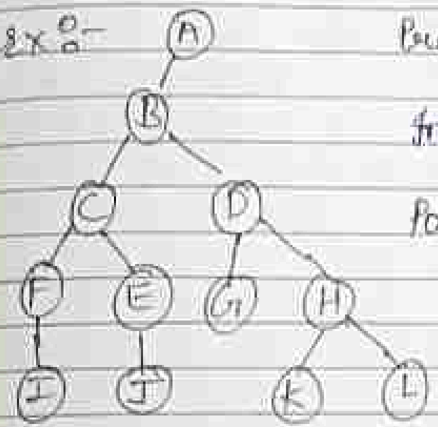


⇒ ABDG CE HIF
⇓
Pre order Traversal

(ii) In order traversal ⇒ LPR ⇒ left subtree than parent and then right tree
Inorder ⇒ DGBAHEICF

(iii) Post order traversal ⇒ LRP
⇒ GDBHJEFCA

Ex :-



Pre :- ABCFIEJ DGHKL

In :- FICEJBGDKHLA

Post :- IFEJCG DKHLA

* Array representation :- If node is at nth position then left will be at 2n+1 and right will be at 2n+2