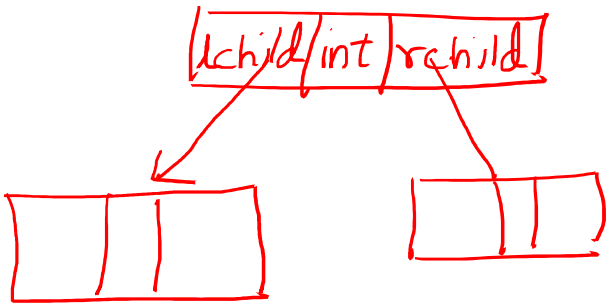
# Binary Tree : Traversal Techniques

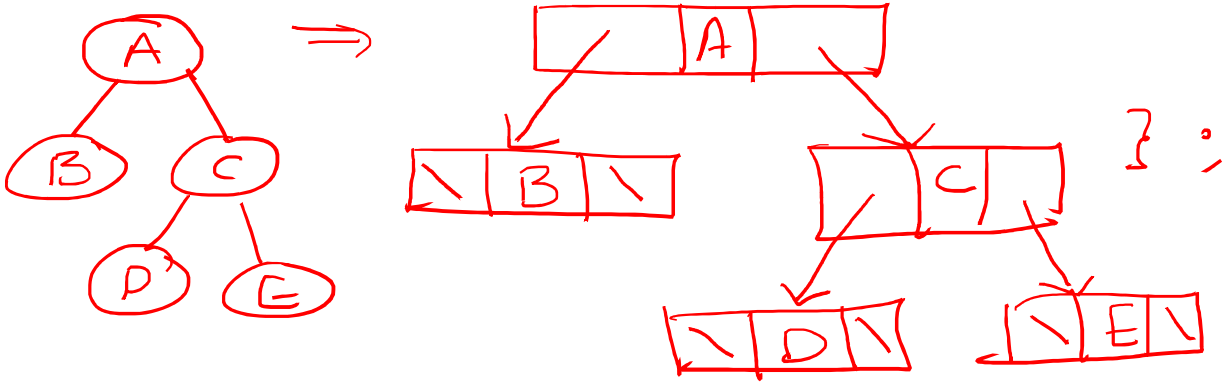
Dec. 04th, 2021

Lecture-18

(2) linked list representation of BT

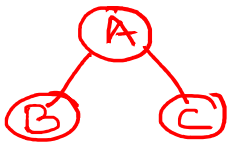



```
class btree{ int data;

btree *lchild;
btree *rchild;

public;
=
=
};
```



# Traversal Techniques:

- Inorder traversal → LVR
- Postorder traversal → LRV
- Preorder Traversal → VLR
- Level-order traversal → level wise display (L → R)

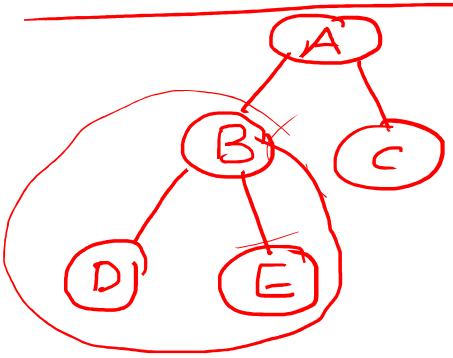



Inorder = B A C

postorder = B C A

preorder = A B C

levelorder = A B C

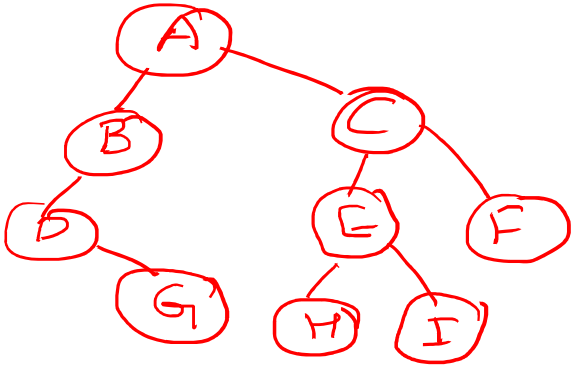



In: D B E A C

post: D E B C A

pre: A B D E C

level: A B C D E

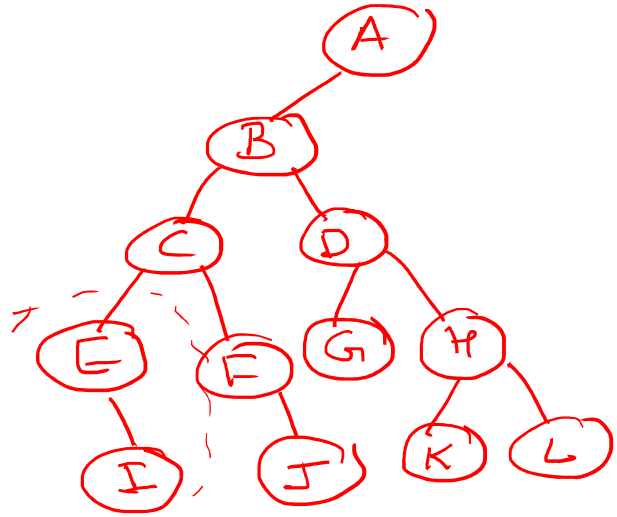



In: D G B A H E I C F

post: G D B H I E F C A

pre: A B D G C E H I F

level: A B C D E F G H I



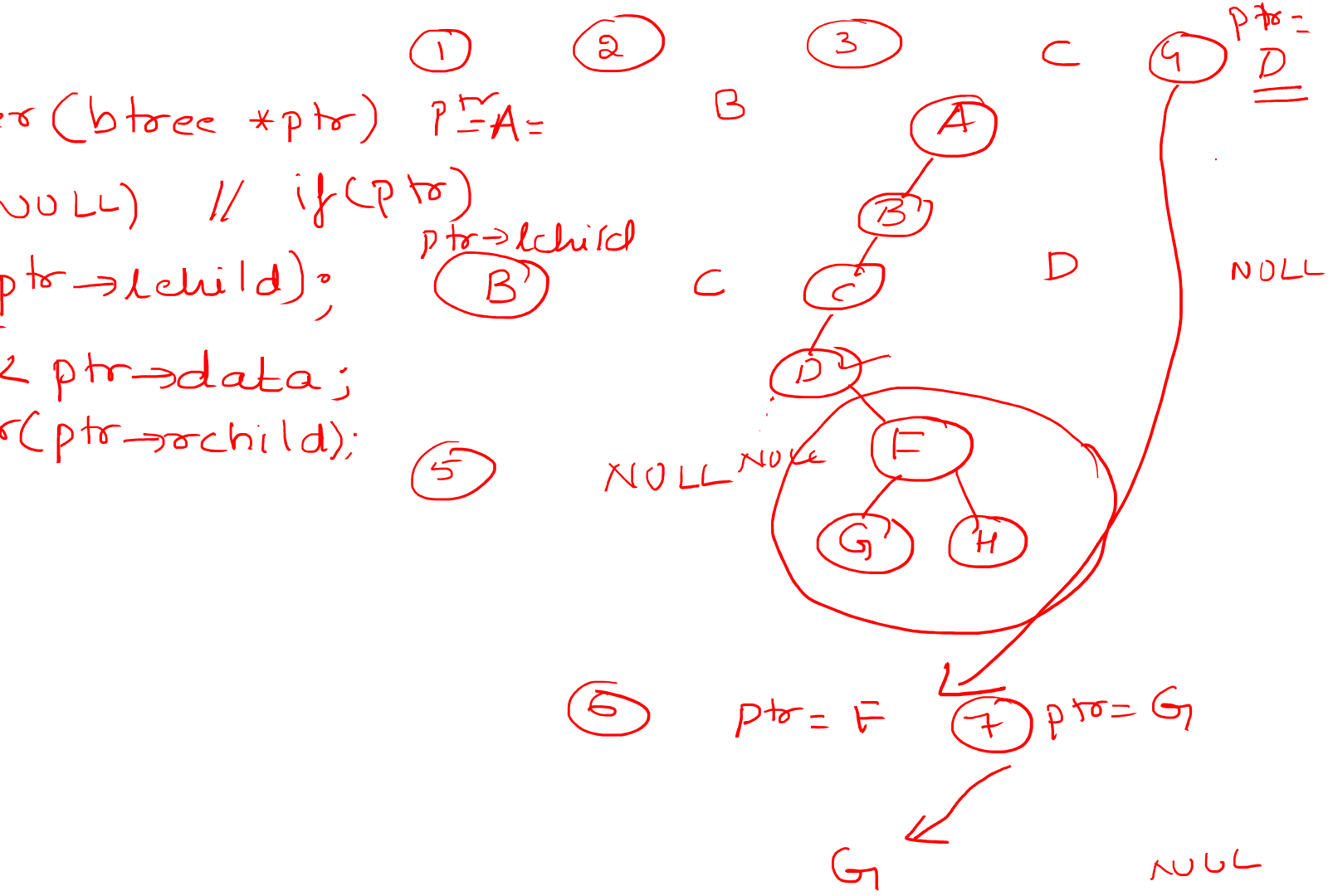

In: E I C F J B G D K H L A

post: I E J F C G K L H D B A

pre: A B C E I F J D G H K L

level: A B C D E F G H I J K L

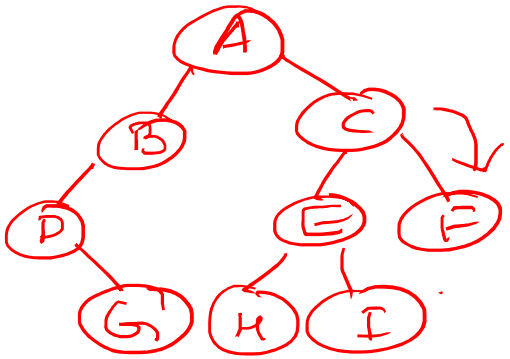
```
void inorder(btree *ptr)
{
    if (ptr != NULL)
    {
        inorder(ptr->lchild);
        cout << ptr->data;
        inorder(ptr->rchild);
    }
}
```




D G

# Recursive inorder

| fn. call | ptr | ptr→child |
|---|---|---|
| 1 | A | |
| 2 | B | |
| 3 | D | cout << p... |
| 4 | NULL | |
| 5 | G | cout << G. |
| 6 | NULL | |
| 7 | NULL | |
| 8 | C | |
| 9 | E | |
| 10 | H | Rout << H |
| 11 | NULL | |

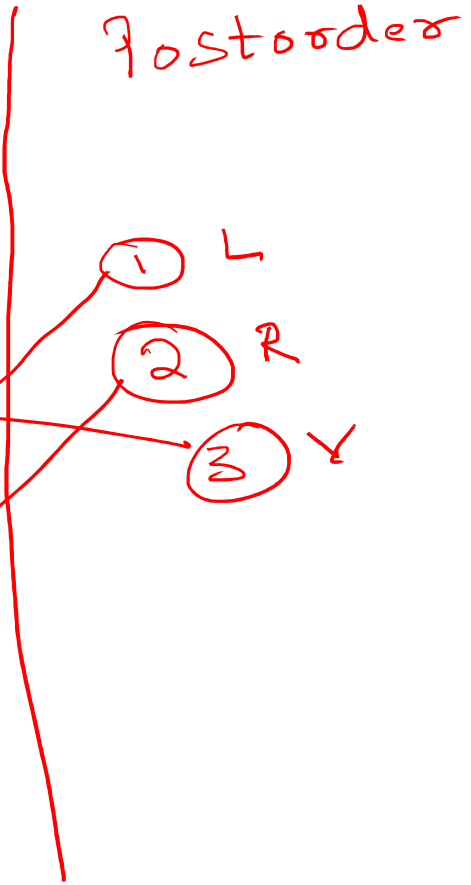




NULL

## Preorder

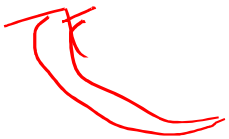
```
void preorder(btree *ptr)
{   if(ptr!=NULL)
    { cout<<ptr->data;
      preorder(ptr->lchild);
      preorder(ptr->rchild);
    }
}
```



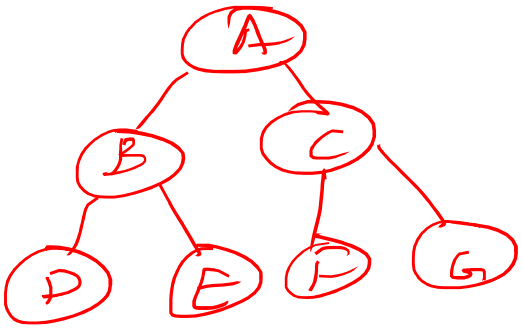
Level-order

0 1 2

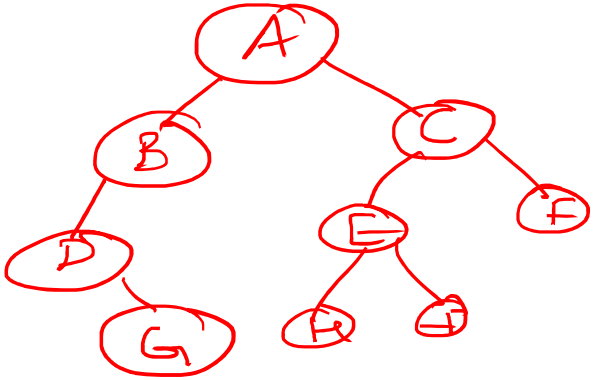
A B C D E F G



Queue




A B C

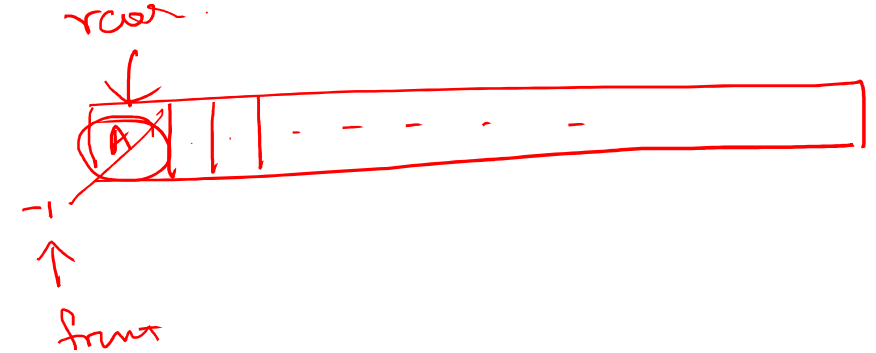



A B C D E F G H I

↑

A B C D E F G H I

```
void levelorder(btree *ptr)
{
    btree *q[30];
    int front=-1, rear=-1;
    if(ptr==NULL) return;
    else
        q[++rear] = ptr;   // root address
    while(1)
    {
        ptr = q[++front];
        if(front==rear)
        { cout<<ptr->data;
          if(ptr->lchild)
              q[++rear] = ptr->lchild;
          if(ptr->rchild)
              q[++rear] = ptr->rchild; }
        else
            break;
    }
```



# Level order traversal

```
void btree::level_order()
{   int f=-1,rear=-1;
    btree *q[10], *ptr=root;
   if(ptr==NULL) return;
   q[++rear]=ptr;//f=-1,rear=0
   cout<<"In level order:";
```

```
for(;;)
   {   if(f!=rear)
       { ptr =q[++f];
         cout<<ptr->data<<" ";
         if(ptr->lchild)
              q[++rear]=ptr->lchild;
         if(ptr->rchild)
             q[++rear] = ptr->rchild;
        }
        else   //Q is empty
           break;
   }
}
```

or

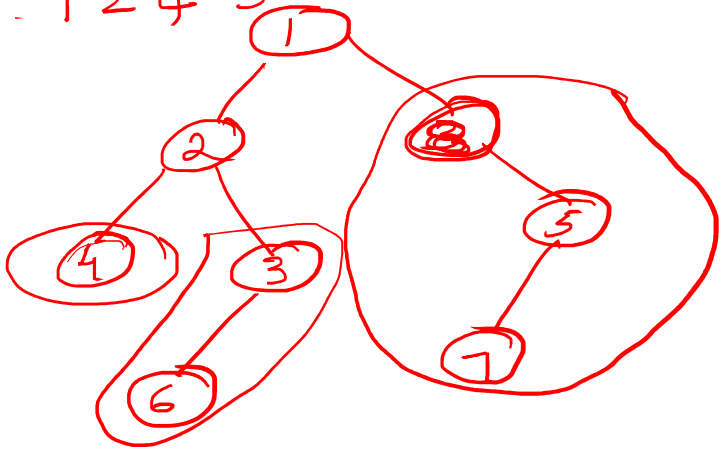
```
while(f!=rear)
      { ptr =q[++f];
        cout<<ptr->data<<" ";
        if(ptr->lchild)
             q[++rear]=ptr->lchild;
        if(ptr->rchild)
             q[++rear] = ptr->rchild;
      }
}
```

# Iterative inorder traversal:logic

Preorder

S: 1 2 4 3 6 8 5 7

o/p: 1 2 4 3 6 8 5 7




void inorder(btree *ptr)

{

1 2 4 3 6 8 5 7

o/p: 4 2 6 3 1 8 7 5

left subtree root

Recursive

if(ptr)

{ inorder(ptr->lchild)

inorder(ptr->rchild);

}

# Iterative inorder traversal

```
void btree:: inorder()
{
   int top=-1;
    btree *s[20], *ptr=root;
    for (;;)
    {
       for(; ptr; ptr=ptr->lchild)
         s[++top] = ptr;
       if(top>=0)
       {  ptr = s[top--];
       cout<<ptr->data<<" ";
       ptr=ptr->rchild;
       }
    }
}
```

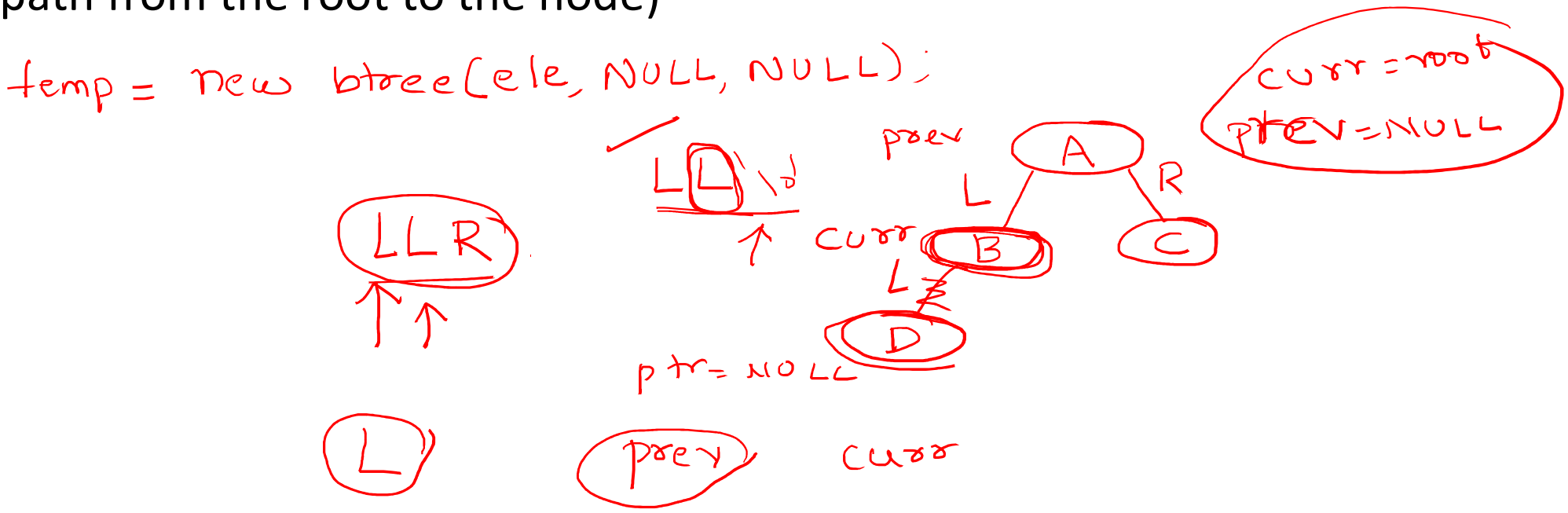
# Iterative Preorder Traversal

```
void btree:: preorder()
{
   int top=-1;
    btree *s[20], *ptr=root;
    for (;;)
    {
       for(; ptr; ptr=ptr->lchild)
         { cout<<ptr->data<<" "; s[++top] = ptr;}

       if(top>=0)
         ptr = s[top--];
    else        break;
       ptr=ptr->rchild;
    }
}
```

Try Iterative postorder traversal

# Creating a Binary Tree: Logic

- Tree is created by taking the direction to the node as an input(ie the path from the root to the node)



# Creating a Binary Tree: ~~Logic~~ Function.

```
void btree::create( )
{   btree *prev, *curr, *temp;
    int i=0, n; int ele; char dir[20];
    cout<<"No. of nodes: "; cin>>n;
   for(i=0;i<n;i++)
   {  cout<<"Enter the key value: "
       cin>>ele;
       temp = new btree(ele, NULL, NULL);
       if(root==NULL)
                root=temp;

       else
```

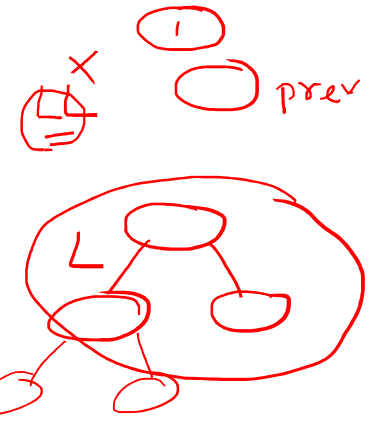
# Creating a Binary Tree: function

```
{ cout<<"Input the direction (uppercase):\n";
  cin>>dir;     //
   curr=Root;   prev=NULL
  for(i=0; dir[i]!='\0' && curr!=NULL; i++)
  {   prev=curr;
      if(dir[i]=='L')
                curr=curr->lchild;
      else
          dir[i] curr=curr->rchild;
  }
```

n = 4

i = 0

i = 1

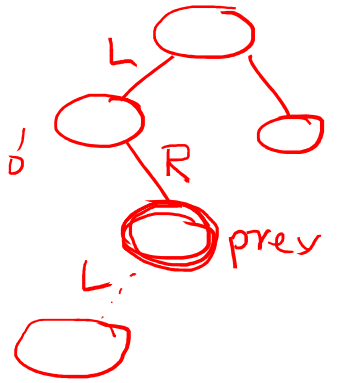


# Creating a Binary Tree: function

```
if(curr != NULL || dir[i] != '\0')
{ cout << "Insertion not possible: error\n";
  delete temp;
}
else
{ if(dir[i-1] == 'L')
    prev->lchild = temp;
  else
    prev->rchild = temp;
}
}
```

dir = L

LRLL'\0'





prev = root

curr = NULL

LRR'\0'

# Creating a Binary Tree: recursive function

```
btree *insert(btree *ptr, char *c)
{   if(ptr==NULL && *c=='\0')
      {   create temp & return its address
      }
    if(*c=='L')
        ptr->lchild=insert(ptr->lchild, ++c)
    else
        ptr->rchild=insert(ptr->rchild, ++c)
}
```

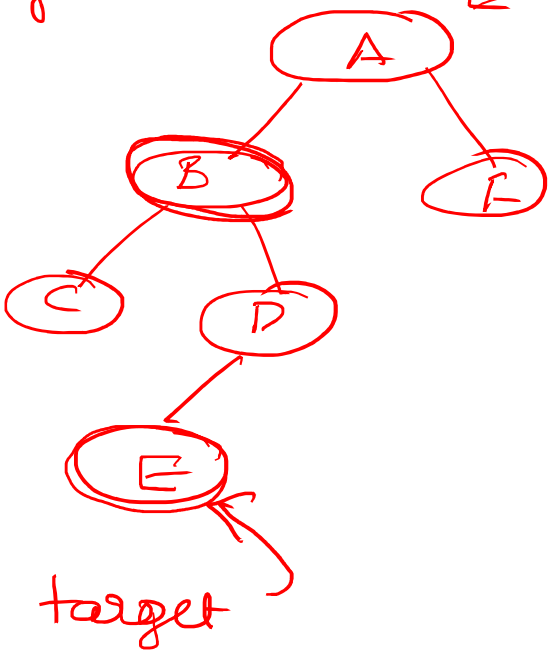
# Copying a Binary Tree: function

```
btree *btree :: copy(btree *ptr)
{
   btree *temp;
   if(ptr)
   {
        temp=new btree;
        if(ptr->leftchild)      temp->leftchild=copy(ptr->leftchild);
        if(ptr->rightchild)     temp->rightchild=copy(ptr->rightchild);
        temp->data=ptr->data;
        return(temp);
   }
   return(NULL);
}
```

# Copying  a Binary Tree: Explanation

# Print Ancestors

```
//If target is present in tree, then prints the ancestors and returns true,
otherwise returns false.

int btree::printAncestors(btree *root, int target)
{
 /* base cases */
 if (root == NULL)     return(0);

 if (root->data == target)      return(1);

 /* If target is present in either left or right subtree of this node,
    then print this node */
 if ( printAncestors(root->leftchild, target) ||
      printAncestors(root->rightchild, target) )
 {
  cout << root->data << " ";
  return 1;
 }

 /* Else return false */
 return 0;
}
```



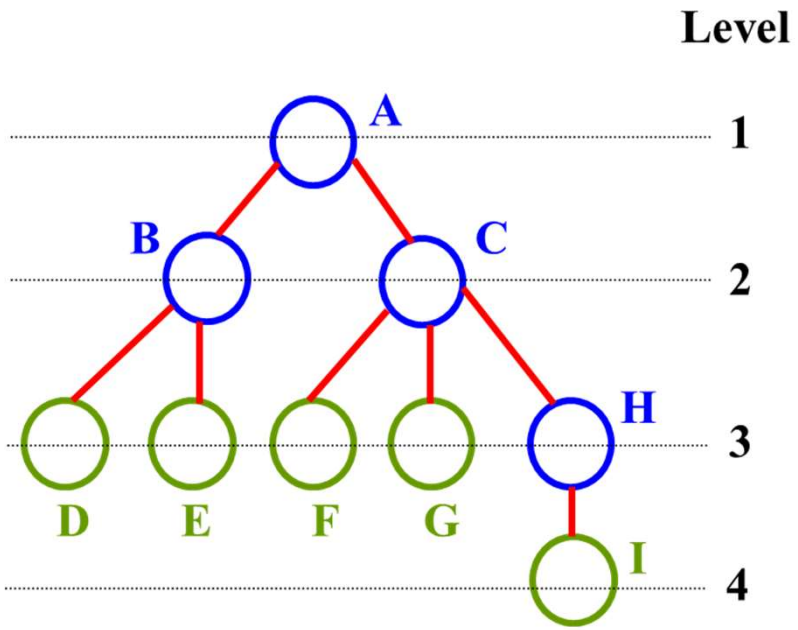
```
btree* btree ::parent(btree *r, int ele)
{
      //root->data==ele then root has no parent so, return NULL
      if(r==NULL||r->data==ele)
            return(NULL);


    if((r->leftchild!=NULL && r->leftchild->data==ele) ||
               (r->rightchild!=NULL && r->rightchild->data==ele))
            return(r);

btree *res=parent(r->leftchild, ele);

if(res!=NULL)   return(res);

res=parent(r->rightchild, ele);

return(res);
}
```

**return the parent node address**

//ele is in leftchild of r

//ele is in rightchild of r

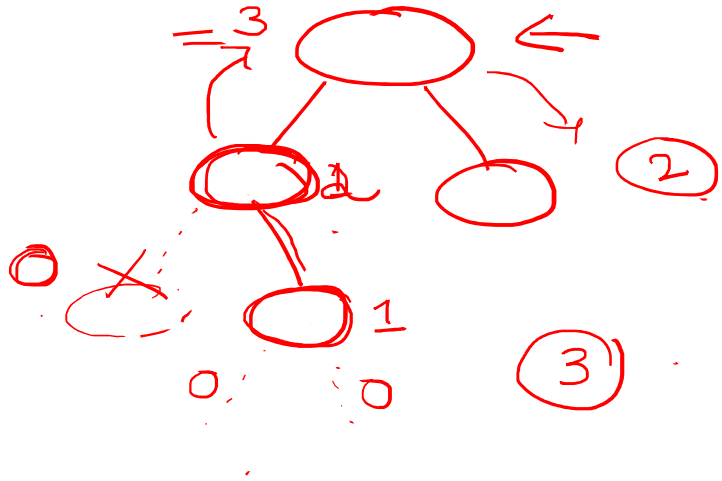
# Height of a tree




The *height (depth)* of the tree is *4*

*Height* (or *depth*): the maximum level of any node in the tree

# Height of a tree



```
int max(int a,int b)
{
        int m=(a>b)?a:b;
        return(m);
}
int btree::FindHeight(btree *r)
{
        if(root==NULL)          return(0);
        return( max(FindHeight(r->leftchild), FindHeight(r->rightchild))  +1);
}
```

- Construct a binary tree using inorder and level order traversal given below.

Inorder Traversal: 3, 4, 2, 1, 5, 8, 9

Level Order Traversal: 1, 3, 9, 2,5, 4, 8

LVR