In [3]:
```
# Task-1

class Tree:
    def __init__(self,data=None):
        self.data=data
        self.left,self.right=None,None

def Height(imp):

    if imp==None:
        return 0

    return 1 + max(Height(imp.left),Height(imp.right))


imp=Tree(1)
imp.left=Tree(2)
imp.right=Tree(3)
imp.left.left=Tree(4)
imp.left.right=Tree(5)
imp.right.left=Tree(6)
imp.right.right=Tree(7)

print('The height of tree is',Height(imp))
```

```
The height of tree is 3
```

In [1]:
```
# Task-2

class Tree(object):
    def __init__(self,data):
        self.data=data
        self.left,self.right=None,None

class Level(object):

    def __init__(self,imp):
        self.imp=imp

    def get(self,imp,x,start):
        if imp==None:
            return
        if imp.data==x:
            return start
        return self.get(imp.right,x,start+1) or self.get(imp.left,x,start+1)


imp=Tree(1)
imp.left=Tree(2)
imp.right=Tree(3)
imp.left.left=Tree(4)
imp.left.right=Tree(5)
imp.right.left=Tree(6)
imp.right.right=Tree(7)

x=int(input('Enter the amount: '))
ev=Level(imp)
temp=ev.get(imp,x,0)

if temp!=None:
    print("The level of",x,'is:',temp)
elif temp==None:
    print('Not present')
```

```
Enter the amount: 5
The level of 5 is: 2
```

In [5]:
```
# Task-(3,4,5)


class Tree:

    def __init__(self,val):
        self.val = val
        self.left,self.right=None,None

#====================== (Task-3) ========================

def PreorderTraversal(root):
    temp_1=[]

    Pre(root,temp_1)
    print(temp_1)

def Pre(root,temp_1):

    if root==None:
        return

    temp_1.append(root.val)
    Pre(root.left,temp_1)
    Pre(root.right,temp_1)
    return


#====================== (Task-4) ========================

def InorderTraversal(root):
    temp_2=[]

    In(root,temp_2)
    print(temp_2)

def In(root,temp_2):

    if root==None:
        return

    In(root.left,temp_2)
    temp_2.append(root.val)
    In(root.right,temp_2)
    return


#====================== (Task-5) ========================

def PostorderTraversal(root):
    temp_3=[]

    Post(root,temp_3)
    print(temp_3)

def Post(root,temp_3):
```

```
    if root==None:
        return

    Post(root.left,temp_3)
    Post(root.right,temp_3)
    temp_3.append(root.val)
    return

root=Tree(1)
root.left=Tree(2)
root.right=Tree(3)
root.left.left=Tree(4)
root.left.right=Tree(5)

#======================== (Task-3 Tester) ============================

print("Preorder traversal ")
PreorderTraversal(root)

#======================== (Task-4 Tester) ============================

print("\nInorder traversal ")
InorderTraversal(root)

#======================== (Task-5 Tester) ============================

print("\nPostorder traversal ")
PostorderTraversal(root)
```

```
Preorder traversal
[1, 2, 4, 5, 3]

Inorder traversal
[4, 2, 5, 1, 3]

Postorder traversal
[4, 5, 2, 3, 1]
```

In [8]:
```
# Task-6

class Tree:

    def __init__(self,data):
        self.data=data
        self.left,self.right=None,None

def Matching(temp,imp):

    if imp!=None and temp!=None:
        return ((temp.data==imp.data) and Matching(temp.right,imp.right)
                 and Matching(temp.left,imp.left))

    elif temp==None and imp==None:
        return True

    return False

r_1=Tree(1)
r_1.left=Tree(2)
r_1.right=Tree(3)
r_1.left.left=Tree(4)
r_1.left.right=Tree(5)


r_2=Tree(1)
r_2.left=Tree(2)
r_2.right=Tree(3)
r_2.left.left=Tree(4)
r_2.left.right=Tree(5)

if Matching(r_1,r_2)==False:
    print ("The trees are not not same")
elif Matching(r_1,r_2)==True:
    print ("The trees are exactly same")
```

```
The trees are exactly same
```

In [1]:
```
# Task-7

class Tree:
    def __init__ (self,data):
        self.data=data
        self.temp,self.left,self.right=None,None,None
def pre(r):
    if r==None:
        return
    print(r.data, end="=>[ ")
    if r.left:
        print(r.left.data,end=', ')
    else:
        print('*',end=', ')

    if r.right:
        print(r.right.data,end=', ')
    else:
        print('*',end=', ')
    if r.temp:
        print(r.temp.data,'] ')
    else:
        print('*','] ')


    pre(r.left)
    pre(r.right)

def clone(r):
    imp={}
    copy(r,imp)
    upd(r,imp)
    return imp[r]


def upd(r,temp):
    if temp.get(r)==None:
        return

    temp.get(r).temp=temp.get(r.temp)
    upd(r.left,temp)
    upd(r.right,temp)

def copy(r,ev):
    if r==None:
        return

    ev[r]=Tree(r.data)
    ev[r].left=copy(r.left,ev)
    ev[r].right=copy(r.right,ev)
    return ev[r]

r=Tree(1)
r.left=Tree(2)
r.right=Tree(3)
r.left.left=Tree(4)
```

```
r.left.right=Tree(5)
r.right.left=Tree(6)


r.left.left.temp=r.right
r.left.right.temp=r
r.right.left.temp=r.left.left
r.temp=r.left

print("Preorder traversal of the given tree:\n")
pre(r)
clone=clone(r)
r.left.right.data=99
print("\nPreorder traversal after changing the data:\n")
pre(r)
print("\nPreorder traversal of the new tree remains same as the")
print('given tree even after changing data of the given tree\n')
pre(clone)
```

```
Preorder traversal of the given tree:

1=>[ 2, 3, 2 ]
2=>[ 4, 5, * ]
4=>[ *, *, 3 ]
5=>[ *, *, 1 ]
3=>[ 6, *, * ]
6=>[ *, *, 4 ]

Preorder traversal after changing the data:

1=>[ 2, 3, 2 ]
2=>[ 4, 99, * ]
4=>[ *, *, 3 ]
99=>[ *, *, 1 ]
3=>[ 6, *, * ]
6=>[ *, *, 4 ]

Preorder traversal of the new tree remains same as the
given tree even after changing data of the given tree

1=>[ 2, 3, 2 ]
2=>[ 4, 5, * ]
4=>[ *, *, 3 ]
5=>[ *, *, 1 ]
3=>[ 6, *, * ]
6=>[ *, *, 4 ]
```

Ans to the Q: No: 8