CSE 1201

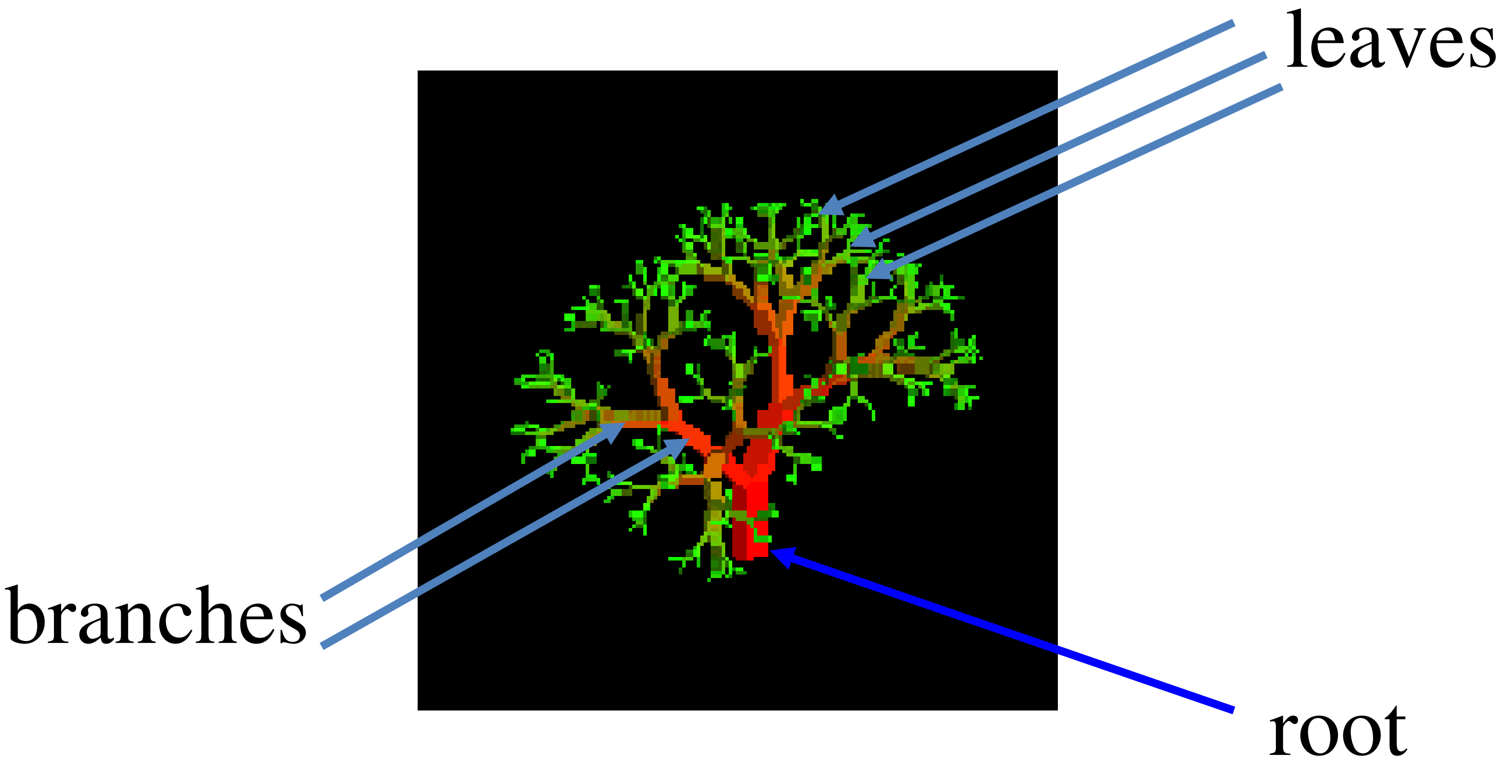
Trees and Binary Trees

Md. Shahid Uz Zaman
Dept. of CSE, RUET

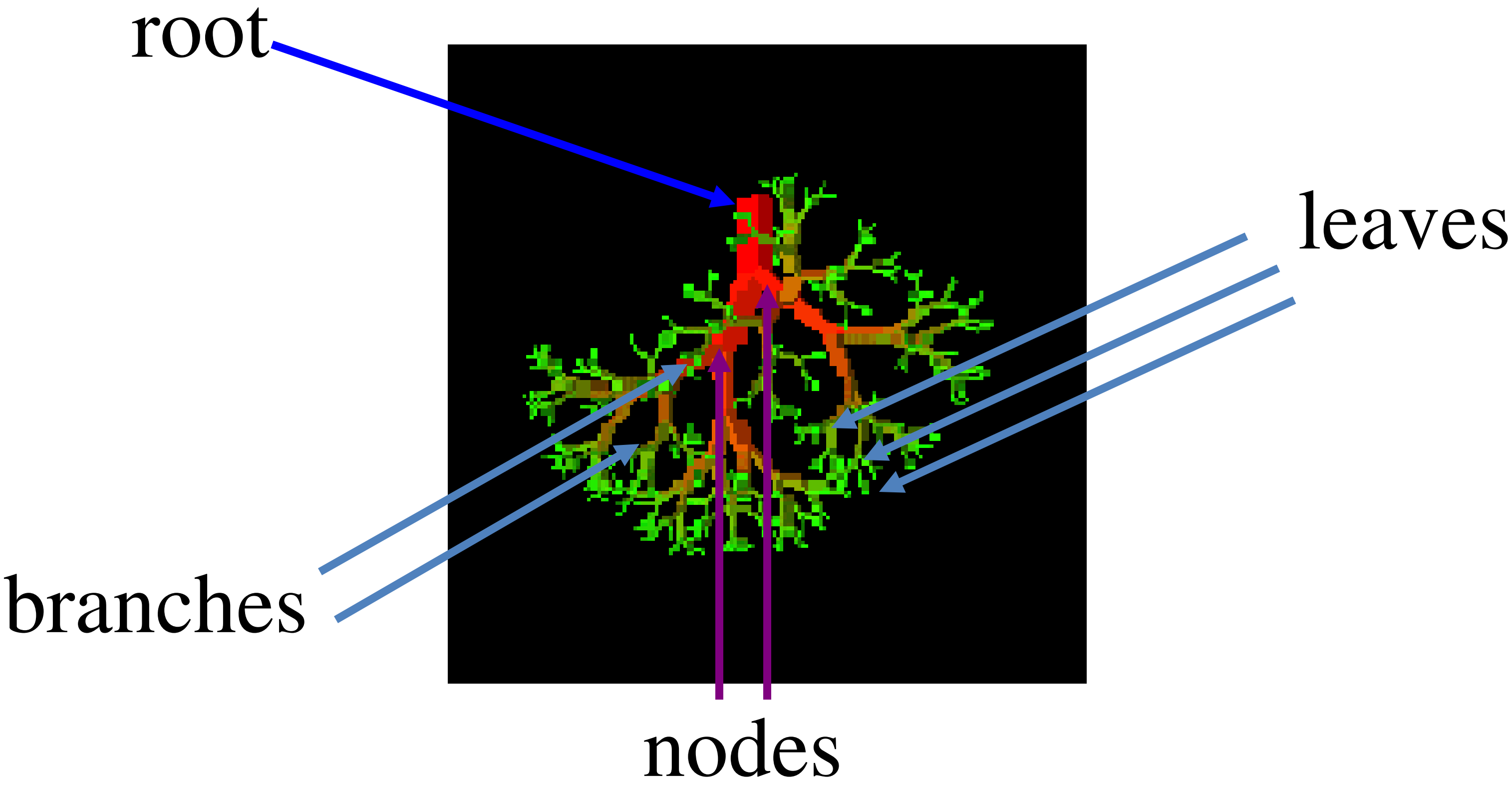
Basics of Tree

- Tree
  - Basic Terminologies
- Binary Tree
  - Binary Tree – Representation

Nature View of a Tree



Computer Scientist's View



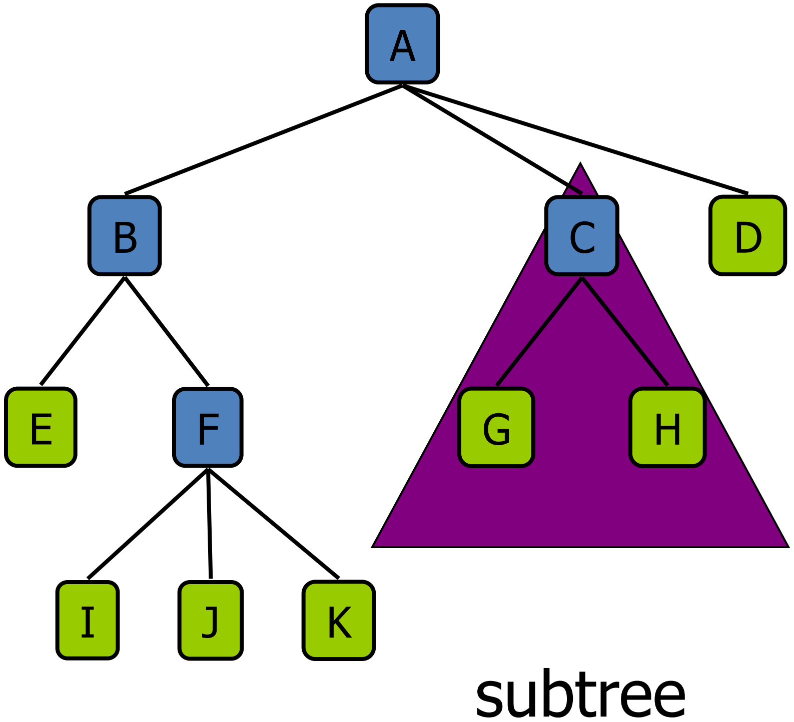
TREES

- Trees are one of the important non- Linear data structure.
- A tree is a Multilevel data structure that represent a hierarchical relationship between the Set of individual elements called nodes.
- Each tree structure starts with a node Which is called the root node of the Tree.

Tree Terminology

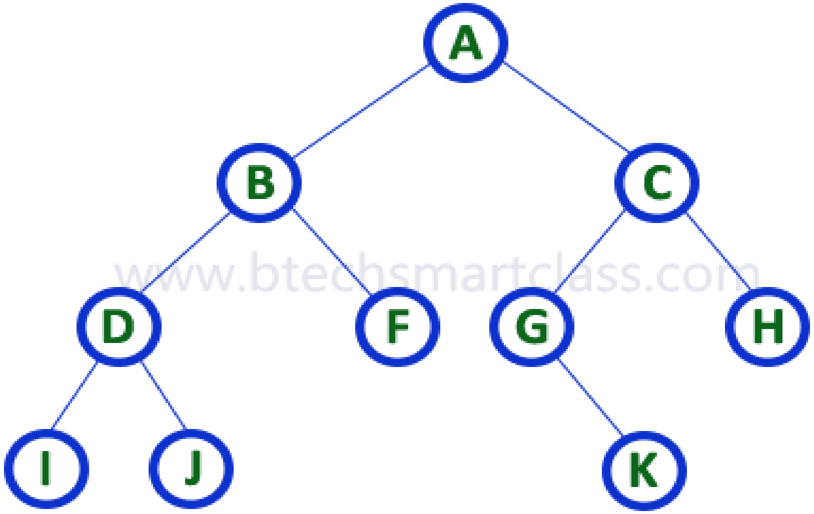
- **Root**: node without parent (A)
- **Siblings**: nodes share the same parent
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)
- **Degree** of a node: the number of its children

- **Subtree**: tree consisting of a node and its descendants



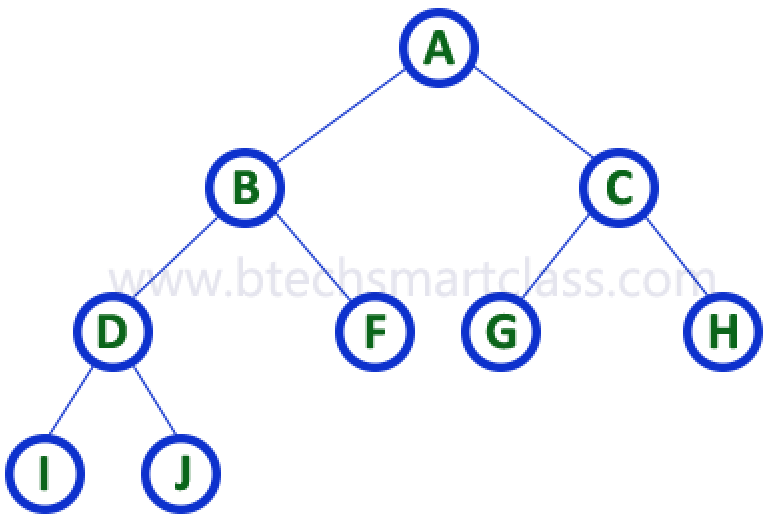
Binary Tree

- A tree in which every node can have a maximum of two children is called as Binary Tree.



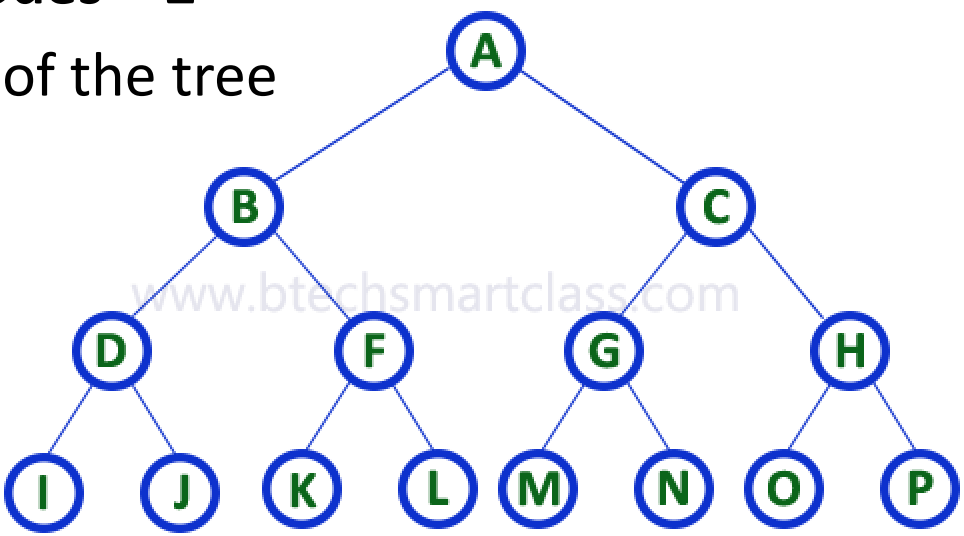
Full Binary Tree

- A binary tree in which every node has either two or zero number of children is called Full Binary Tree



Complete Binary Tree

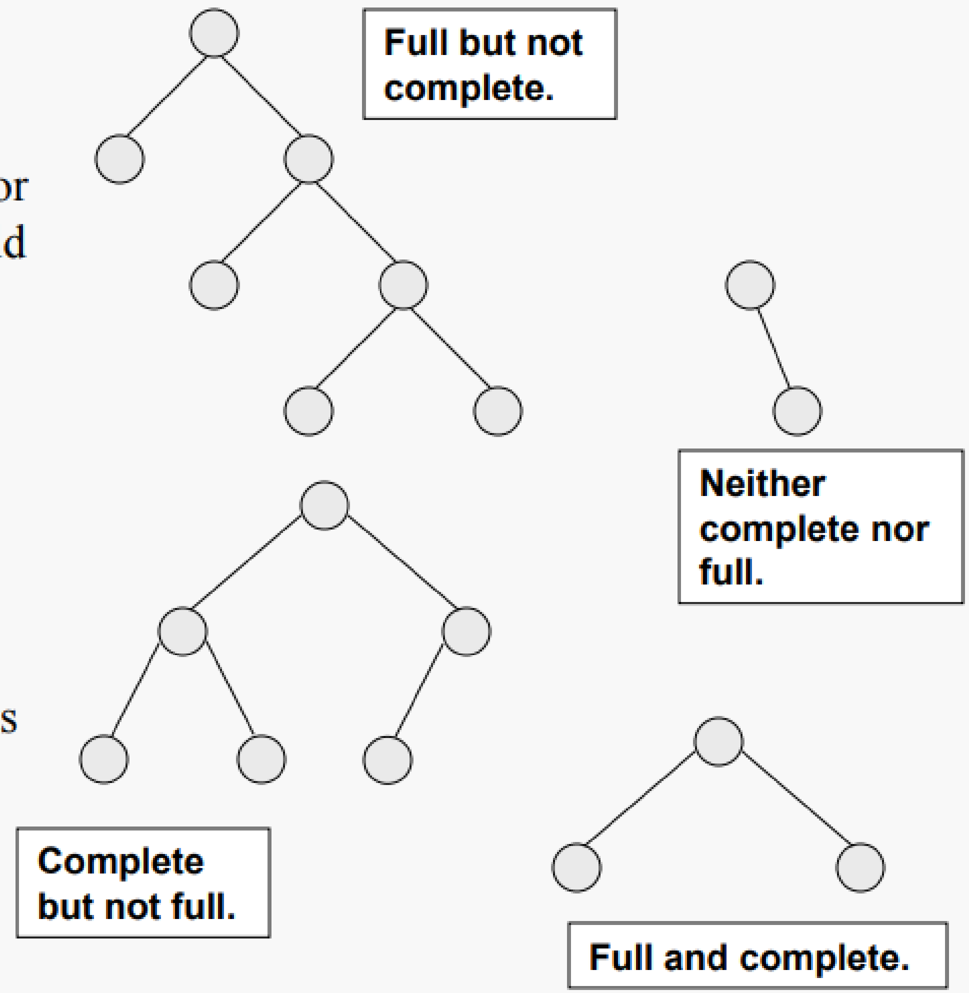
- A binary tree in which every internal node has exactly two children and all leaf nodes are possibly left at same level is called Complete Binary Tree. Complete binary tree is also called as **Perfect Binary Tree**
- **Number of nodes = $2^{d+1} - 1$**
- **Number of leaf nodes = 2^{d}**
- Where, d – Depth of the tree



Complete vs. Full Binary Tree

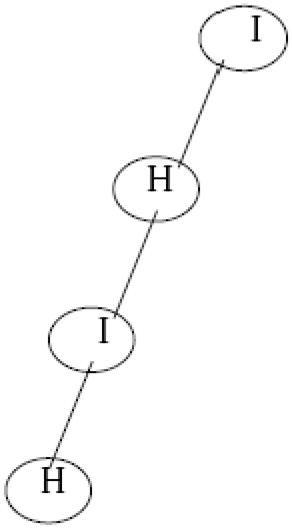
Definition: a binary tree T is *full* if each node is either a leaf or possesses exactly two child nodes.

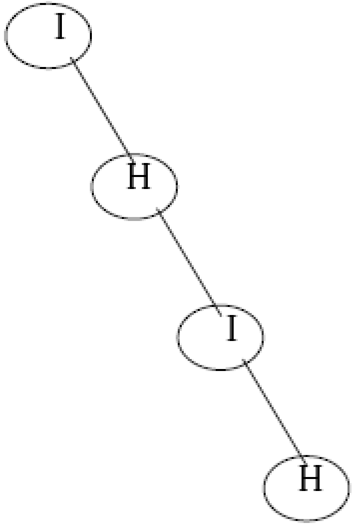
Definition: a binary tree T with n levels is *complete* if all levels except possibly the last are completely full, and the last level has all its nodes to the left side.



Left Skewed and Right Skewed Trees

- Binary tree has only left sub trees - Left Skewed Trees
- Binary tree has only right sub trees - Right Skewed Trees

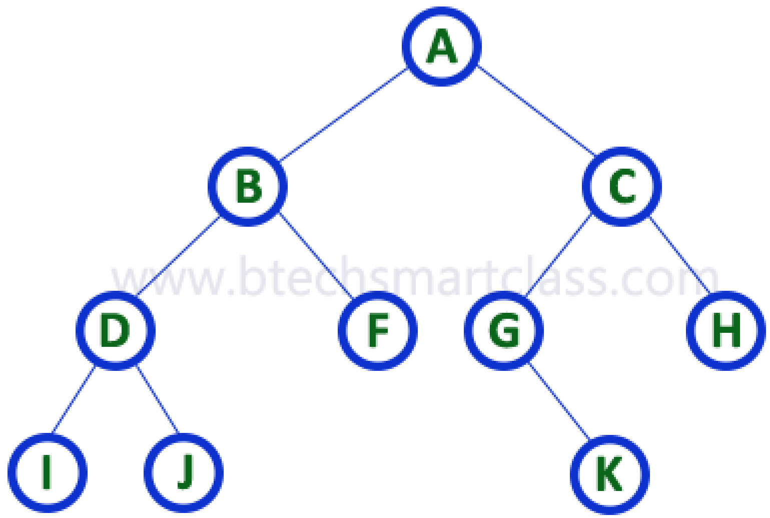




Binary Tree Representation

1. Sequential representation using arrays
2. List representation using Linked list

Sequential representation



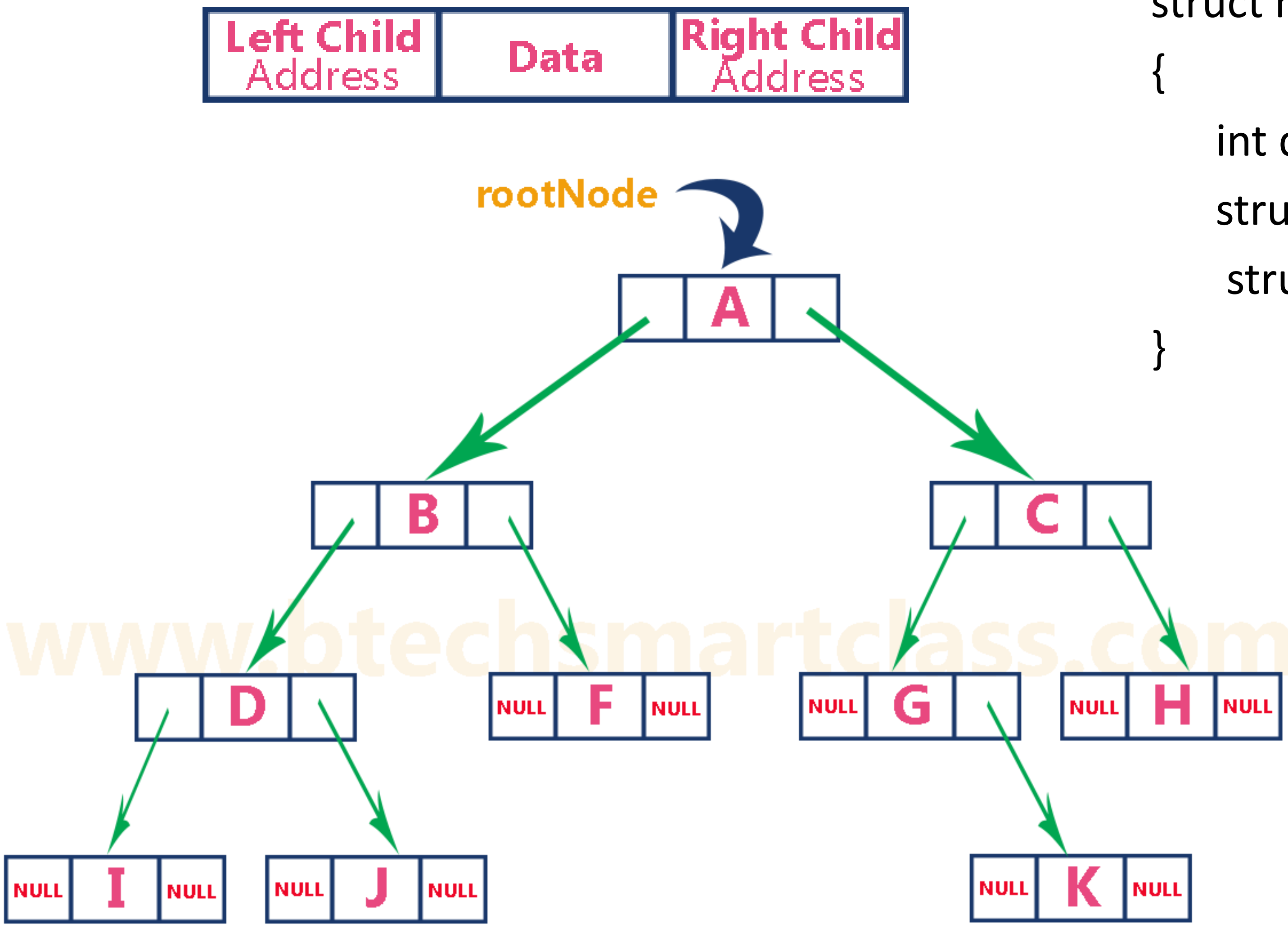
A	B	C	D	F	G	H	I	J	-	-	-	K	-	-
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- To represent a binary tree of depth **‘d'** using array representation, we need one dimensional array with a maximum size of $\mathbf{2^{d+1} - 1}$.

Sequential representation

- Advantages:
  - Direct access to all nodes (Random access)
- Disadvantages:
  - Height of tree should be known
  - Memory may be wasted
  - Insertion and deletion of a node is difficult

List representation




```
struct node
{
    int data;
    struct node *left;
     struct node *rifgt;
}
```

List representation

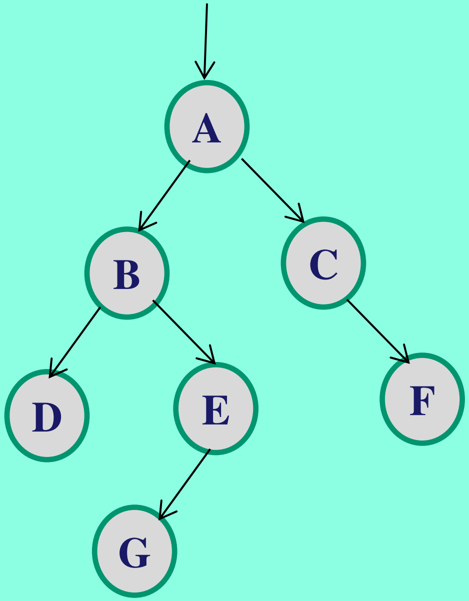
- Advantages:
  - Height of tree need not be known
  - No memory wastage
  - Insertion and deletion of a node is done without affecting other nodes
- Disadvantages:
  - Direct access to node is difficult
  - Additional memory required for storing address of left and right node

Non-Linear Data Structure

- In a non linear data structure , the Elements are not arranged in sequence.
- The data members are arranged in any Manner. The data items are not processed one After another.
- Trees and graphs are examples of non linear data structure.

Binary Tree

Topic 1: Create binary tree from a file

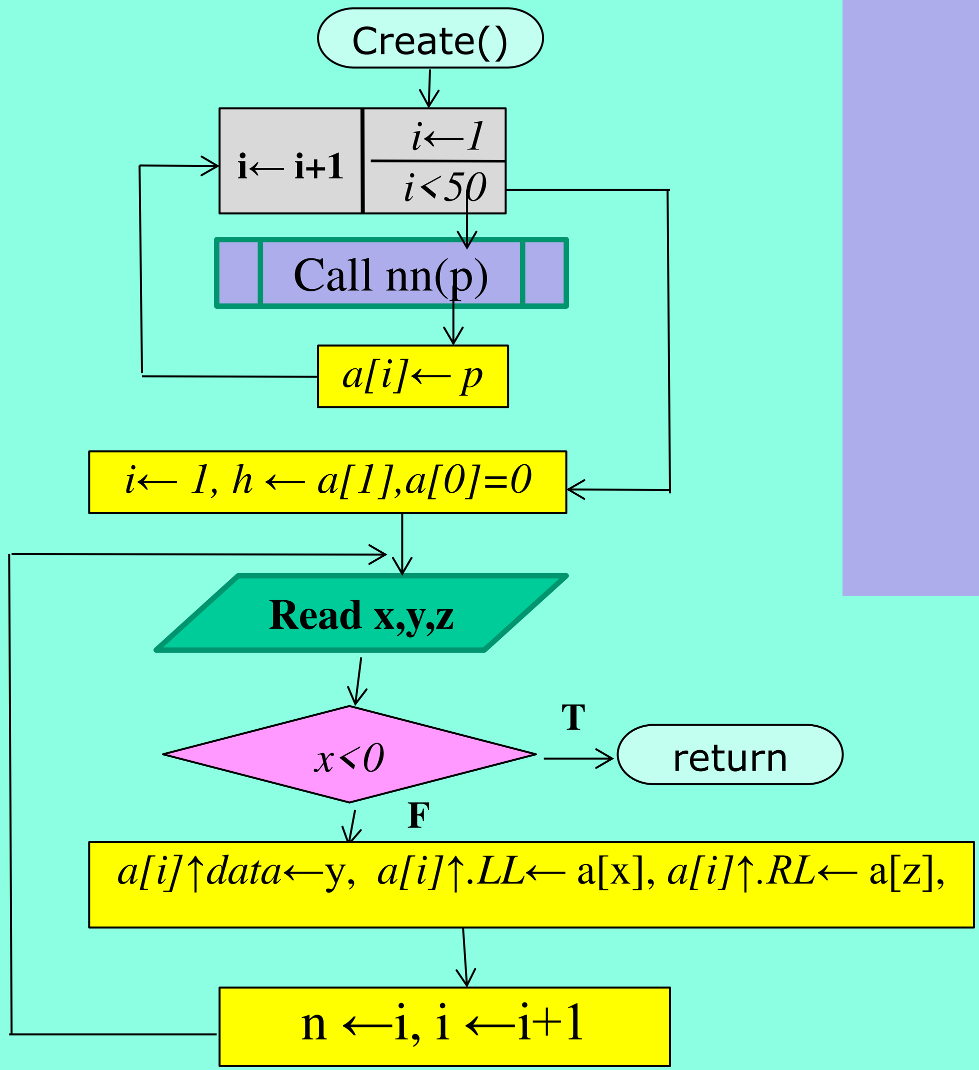



```
2 65 3
4 66 5
0 67 6
0 68 0
7 69 0
0 70 0
0 71 0
```

Text File

```
#include <stdio.h>
struct node{
int data;
struct node *ll;
struct node *rl;
};

struct node *h,*q,*p,*ax[50];
int n;
void main(){
 void create();
 create();
return 0;
}
```

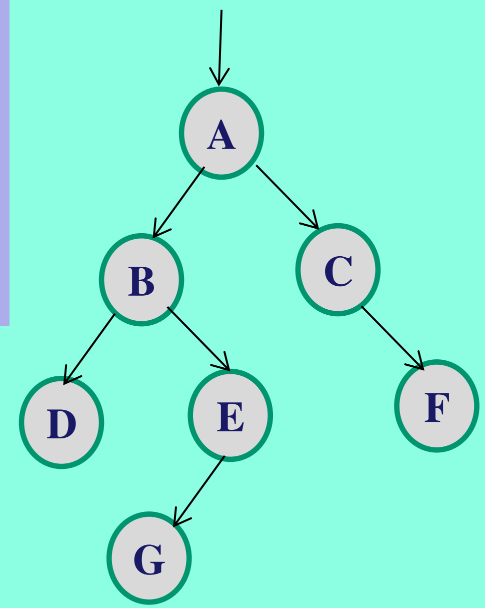
Create a Binary Tree




```
2 65 3
4 66 5
0 67 6
0 68 0
7 69 0
0 70 0
0 71 0
-1 -1 -1
```

Text File

h : root address
LL: Left Link
RL: Right Link
a[] : array of pointers



Binary Tree

Reading & Writing a C++ text file

```
#include<iostream>
#include<fstream>
using namespace std;

int main(){
   ofstream file;
   file.open("F:\\mytext.txt");
   int i=10;
   file<<i;
   file.close();

}
```

```
#include<iostream>
#include<fstream>
using namespace std;

int main(){
   ifstream file;
   file.open("F:\\mytext.txt");
   int x,i,j,ll[50],data[50],rl[50];
   i=1;
   while(file>>ll[i]){
       file>>data[i];
       file>>rl[i];
       i++;
      }
   for(j=0;j<i;j++)
     cout<<ll[j]<<" "<<data[j]<<" "<<rl[j]<<endl;
   file.close();

}
```

Binary Tree

Topic 1: Create binary tree from a file

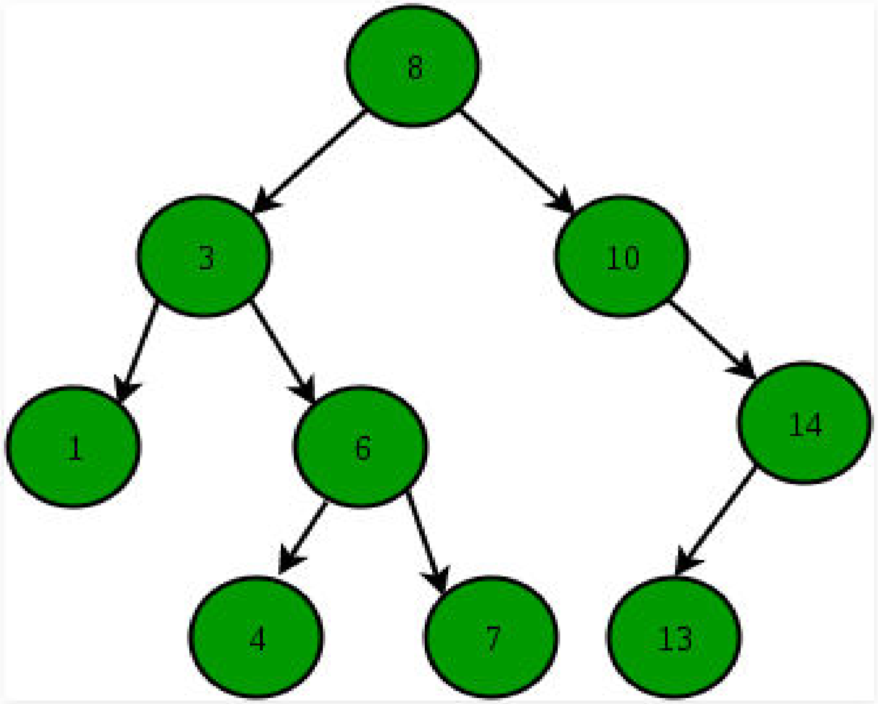
```
#include<iostream>
#include<fstream>
using namespace std;

struct Node{
  struct Node *ll;
  int d;
  struct Node *rl;
};
Node *ax[50];
Node *h;

int main(){
   ifstream file;
   file.open("F:\\mytext.txt");
   int
n,x,i,j,ll[50],data[50],rl[50];
   i=1;
   while(file>>ll[i]){
       file>>data[i];
       file>>rl[i];
       i++;
      }
```

```
     n=i-1;
    for(j=0;j<=n;j++)
      cout<<ll[j]<<" "<<data[j]<<" "<<rl[j]<<endl;
    file.close();

     for(i=1;i<=n;i++){
         ax[i]=new Node();
         cout<<i<<" "<<ax[i]<<endl;
     }
     h=ax[1];
     ax[0]=0;
     for(i=1;i<=n;i++){
       ax[i]->d=data[i];
       ax[i]->ll=ax[ll[i]];
       ax[i]->rl=ax[rl[i]];
     }

     for(i=1;i<=n;i++){
       cout<<ax[i]->ll<<" "<<char(ax[i]->d)<<"
"<<ax[i]->rl<<endl;
     }

}
```

Binary Search Tree

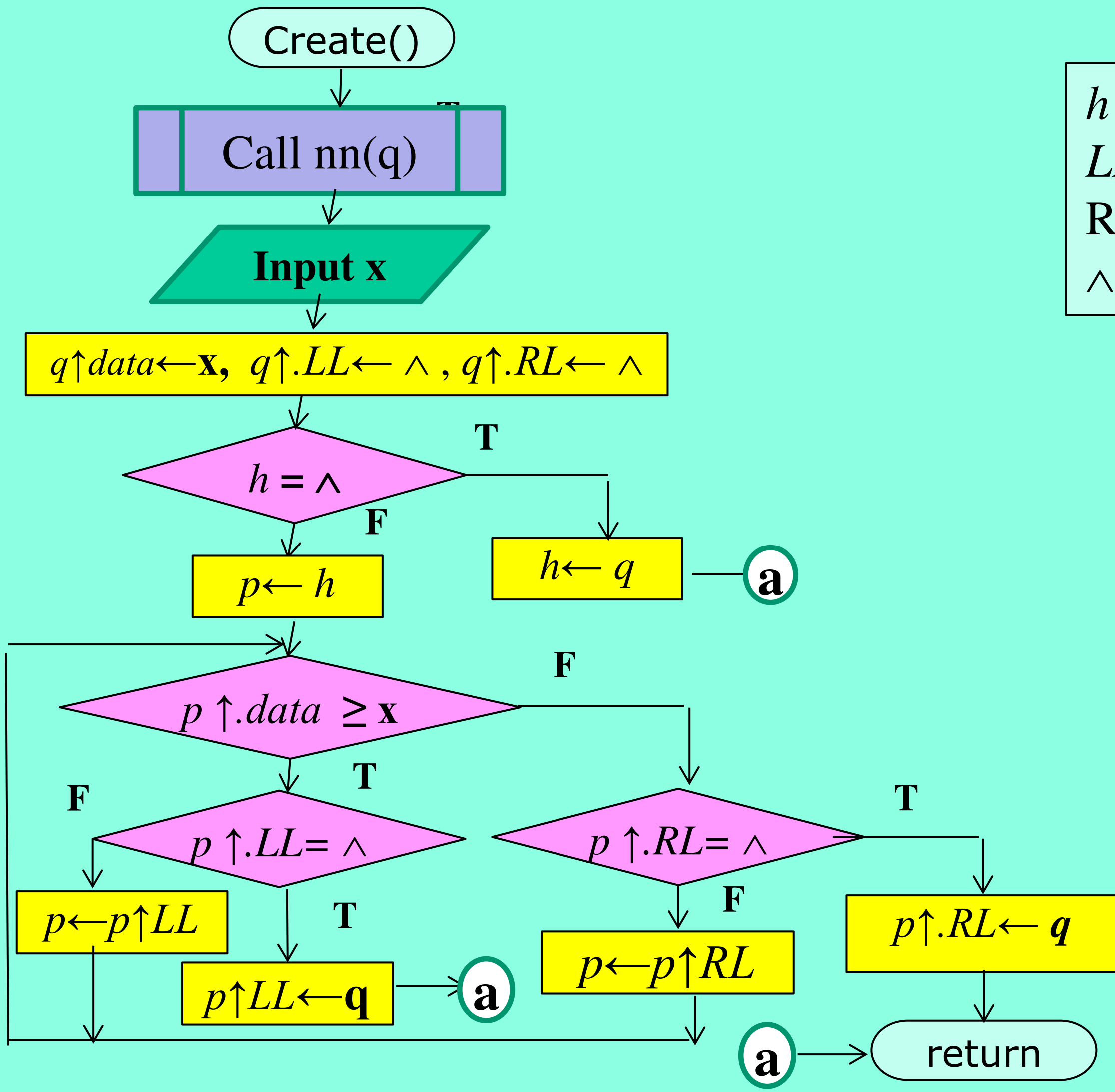
Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

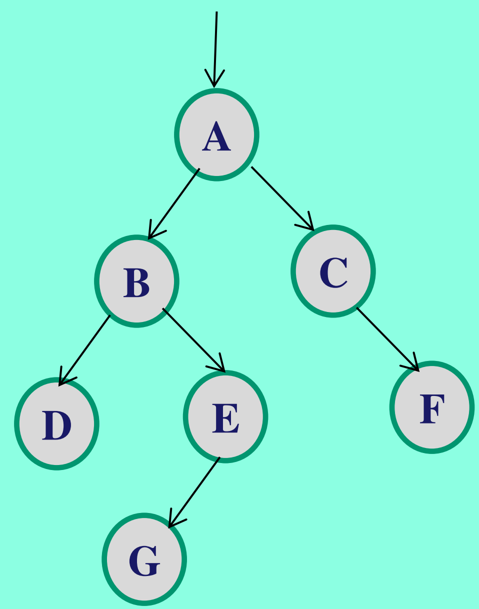


Binary Search Tree

Topic 1: Write an Algorithm to insert a new node in BST

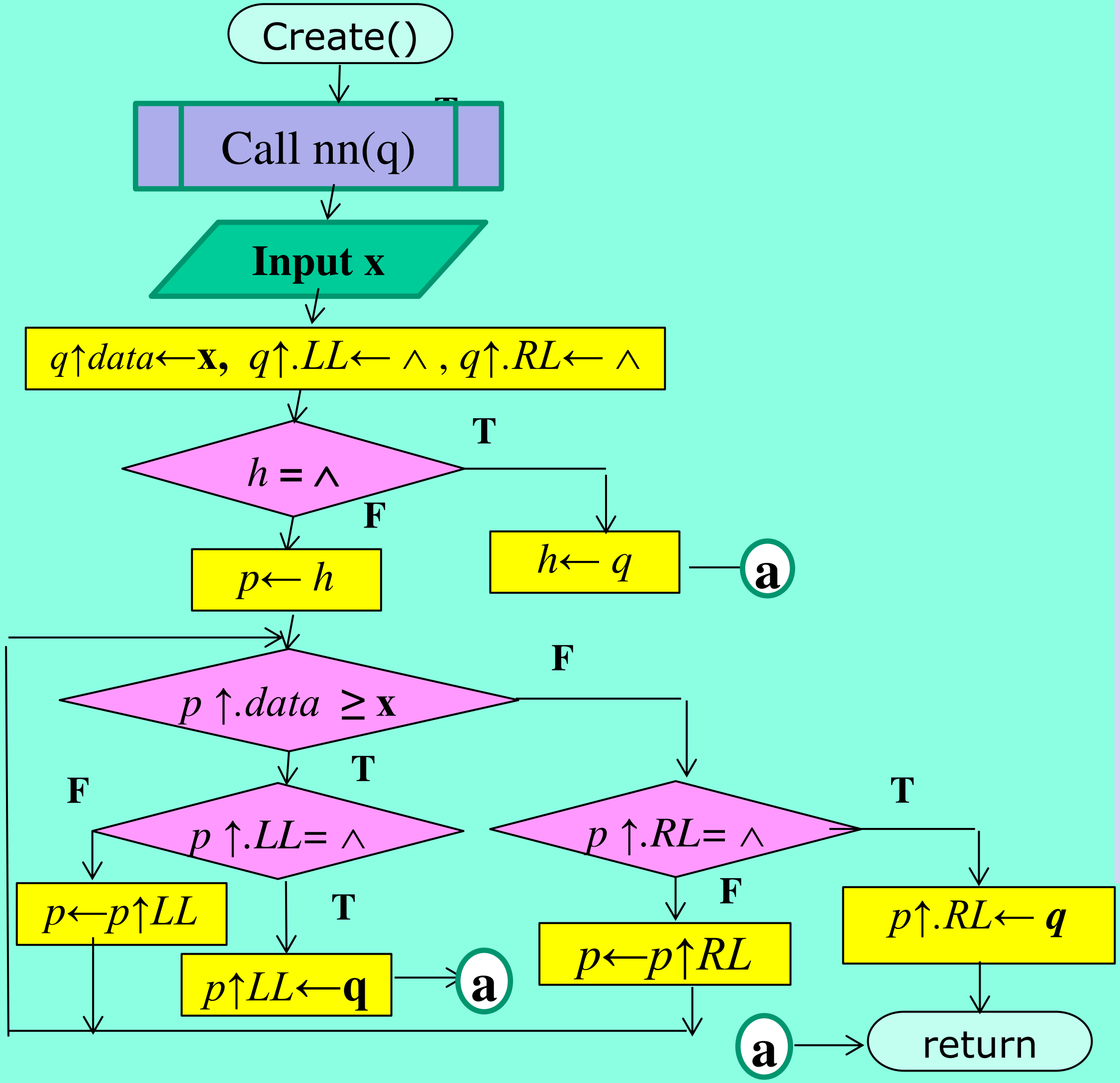


h : root address
LL: Left Link
RL: Right Link
$\wedge$: NULL



Binary Search Tree

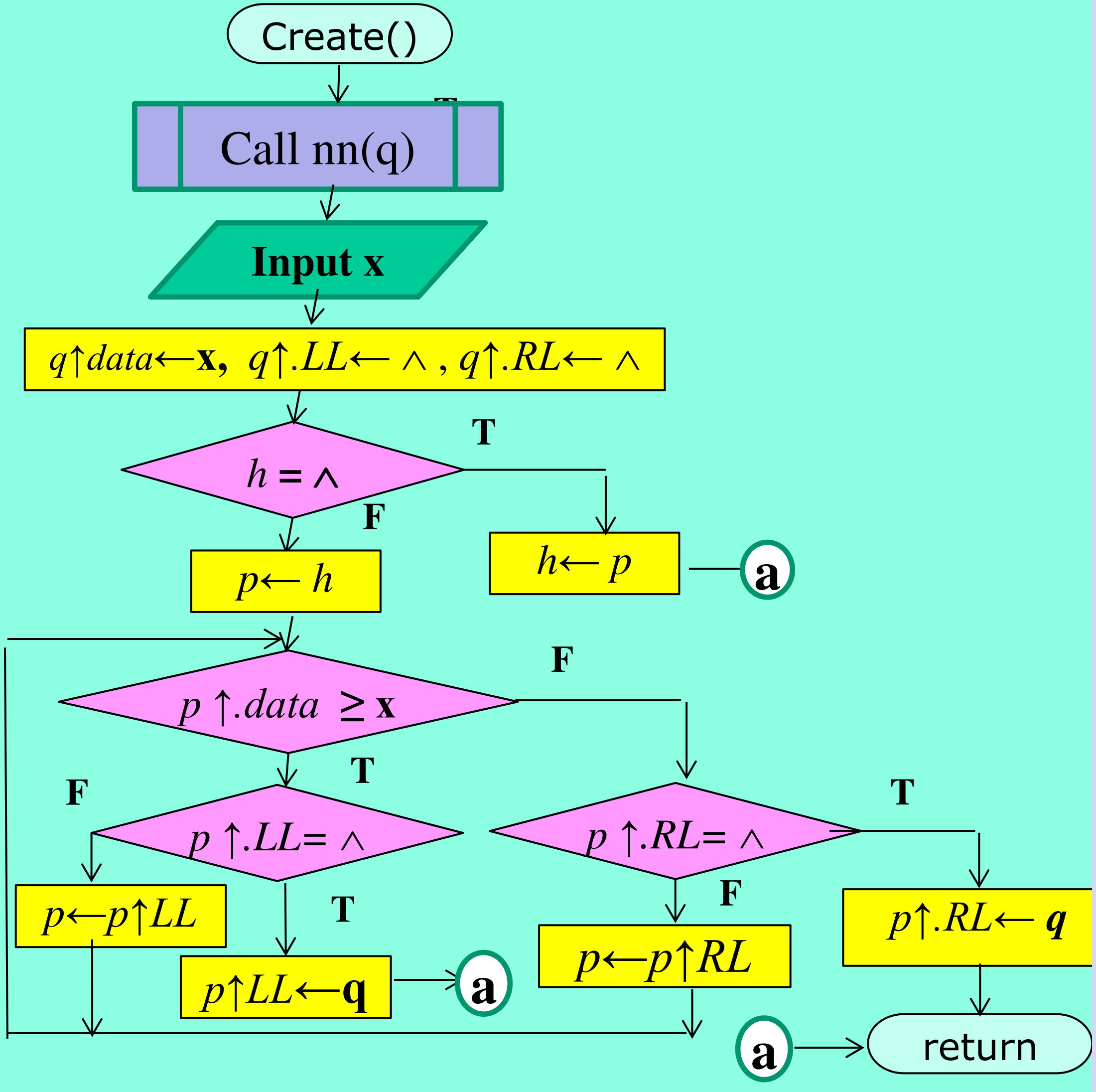
Topic 1: Write an Algorithm to insert a new node in BST




```c
#include <stdio.h>
struct node{
int data;
struct node *ll;
struct node *rl;
};
struct node *h,*q,*p;
int n;
void main(){
 void insert();
 int i,x;
 h=0;
 for(;;){
    printf("Enter DATA:");
    scanf("%d",&x);
    if(x==0) break;
    insert(x);
 }
return 0;
}
```

Binary Search Tree

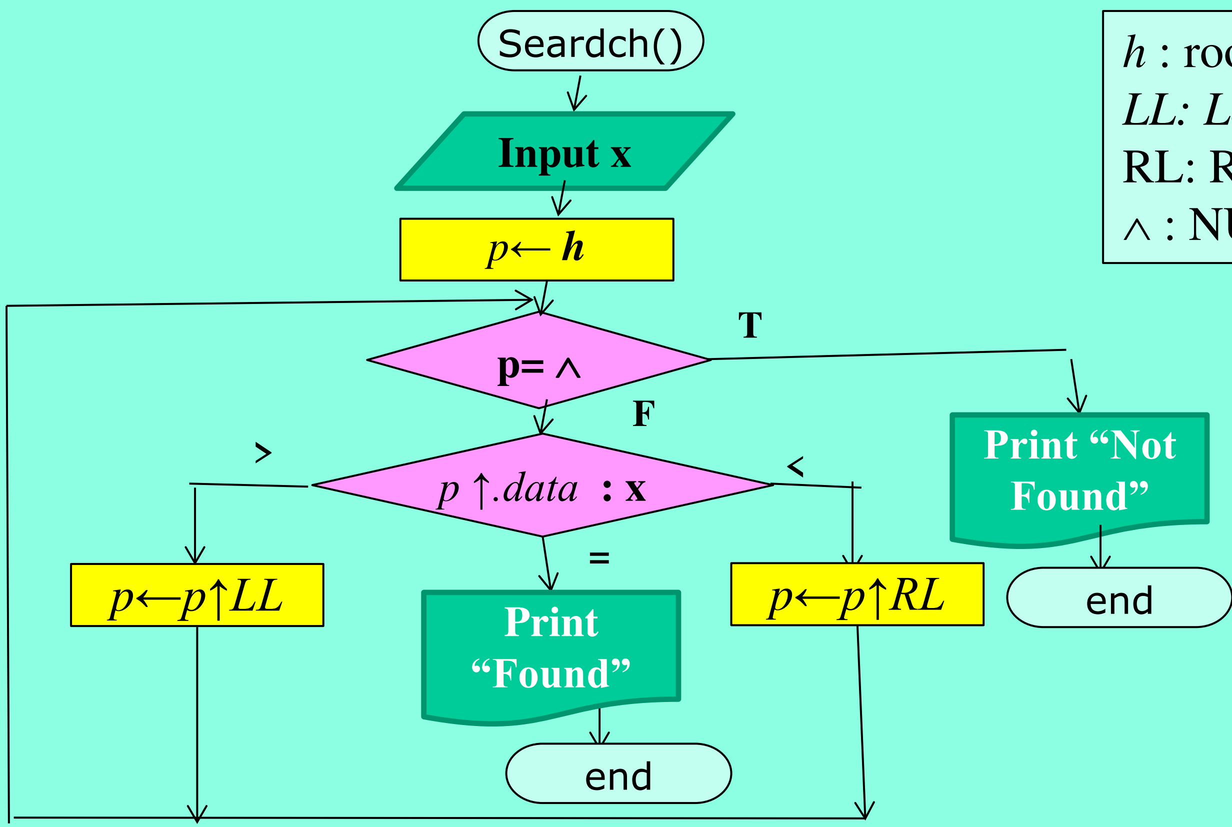
Topic 1: Write an Algorithm to insert a new node in BST.



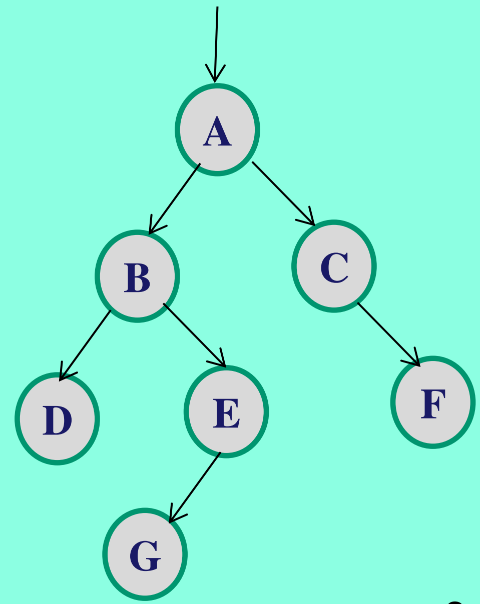

```c
void insert(int x){
  q=(struct node*)(malloc(sizeof(struct node)));
  printf("%X\n",q);
  q->data=x;q->ll=0;q->rl=0;
  if(h==0){  h=q; }
  else{
    p=h;
    for(;;){
    if(p->data>x)
      if(p->ll==0)
        {p->ll=q;
         printf("Node: %X %d %X\n",p->ll,p->data,p->rl);
         break;}
      else
        p=p->ll;
    else
        if(p->rl==0)
        {p->rl=q;
        printf("%X %d %X\n",p->ll,p->data,p->rl);
        break;}
      else
        p=p->rl;
    }
  }
}
```

Binary Search Tree

Topic 1: Write an Algorithm to Search a node in BST



h : root address
LL: Left Link
RL: Right Link
∧ : NULL



Binary Search Tree: (Delete)

When we delete a node, three possibilities arise.

1) *Node to be deleted is leaf:* Simply remove from the tree.

```
       50                            50
    /     \         delete(20)      /   \
   30       70      --------->    30      70
  /  \    /  \                     \    /  \
20   40  60   80                   40  60   80
```

2) *Node to be deleted has only one child:* Copy the child to the node and delete the child

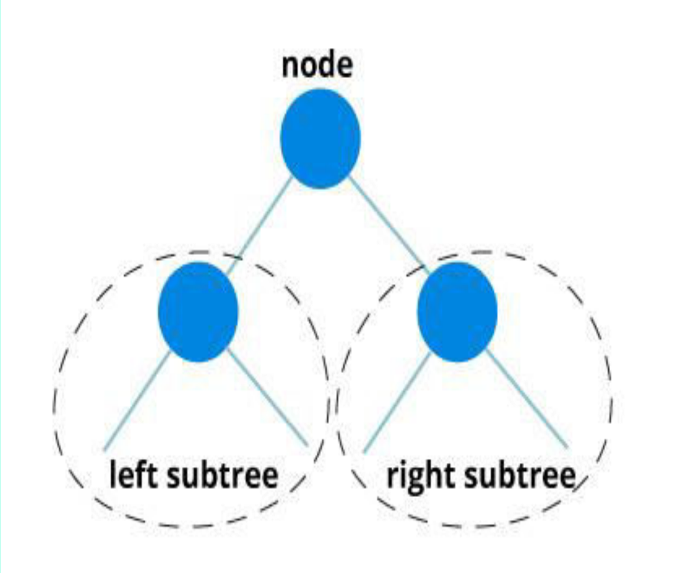
```
       50                            50
    /     \         delete(30)      /   \
   30       70      --------->    40      70
     \    /  \                          /  \
     40  60   80                       60   80
```

3) *Node to be deleted has two children:* Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

```
       50                            60
    /     \         delete(50)      /   \
   40       70      --------->    40     70
          /  \                             \
         60   80                            80
```

Tree Traversing

Traversing a tree means visiting every node in the tree. You might for instance want to add all the values in the tree or find the largest one. For all these operations, you will need to visit each node of the tree.



Inorder traversal (LDR)

1. First, visit all the nodes in the left subtree
2. Then the root node
3. Visit all the nodes in the right subtree

Preorder traversal (DLR)

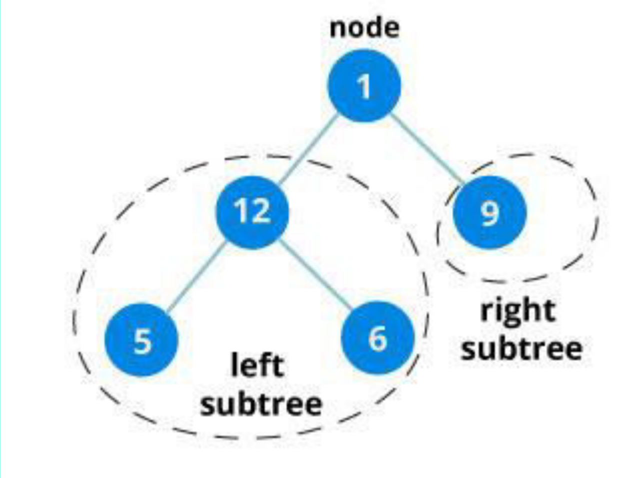
1. Visit root node
2. Visit all the nodes in the left subtree
3. Visit all the nodes in the right subtree

Postorder traversal (LRD)

1. visit all the nodes in the left subtree
2. visit the root node
3. visit all the nodes in the right subtree

Tree Traversing

Traversing a tree means visiting every node in the tree. You might for instance want to add all the values in the tree or find the largest one. For all these operations, you will need to visit each node of the tree.



1. Inorder traversal (LDR)

1.First, visit all the nodes in the left subtree

2.Then the root node

3.Visit all the nodes in the right subtree

2. Preorder traversal (DLR)

1.Visit root node

2.Visit all the nodes in the left subtree

3.Visit all the nodes in the right subtree

3. Postorder traversal (LRD)

1.visit all the nodes in the left subtree

2.visit the root node

3.visit all the nodes in the right subtree

Inorder traversal

5 ->12 ->6 ->1 ->9

Preorder traversal

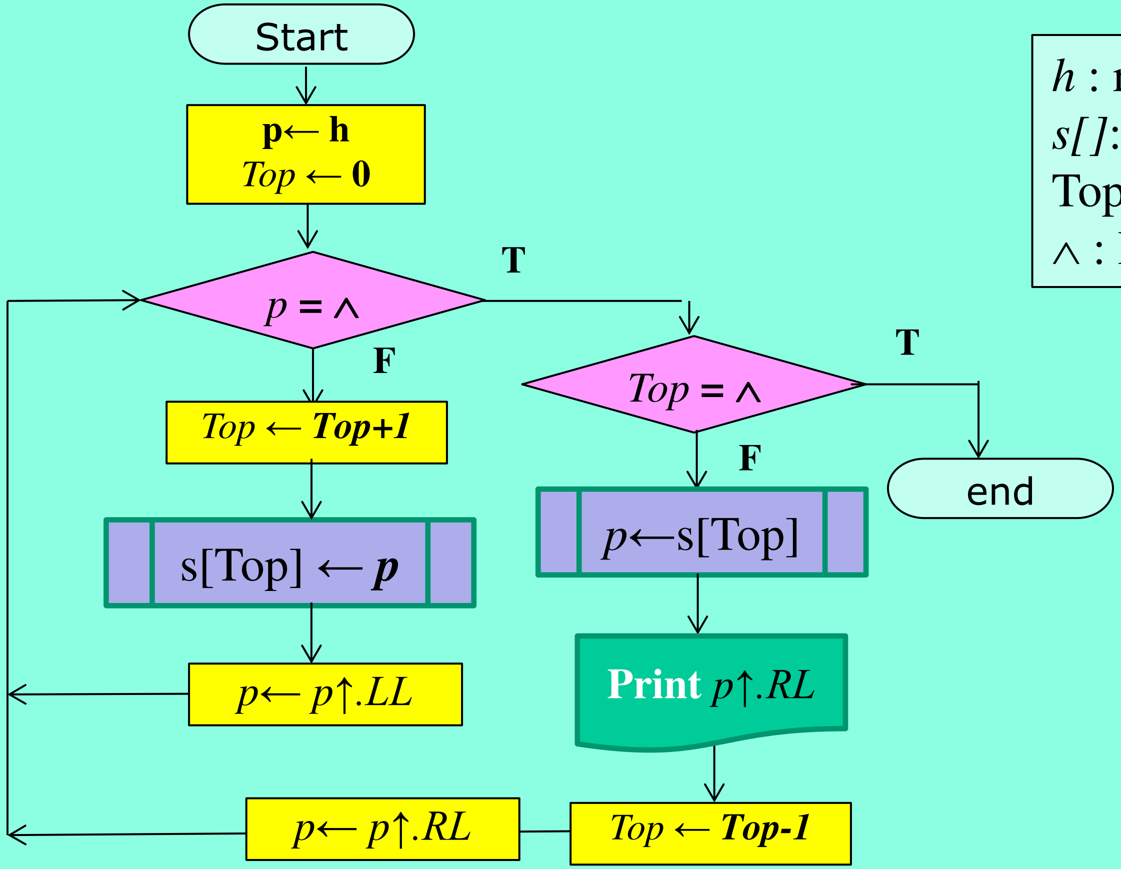
1 ->12 ->5 ->6 ->9

Postorder traversal

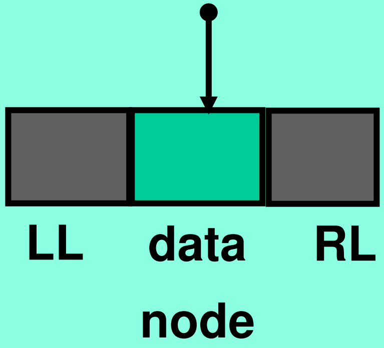
5 ->6 ->12 ->9 ->1

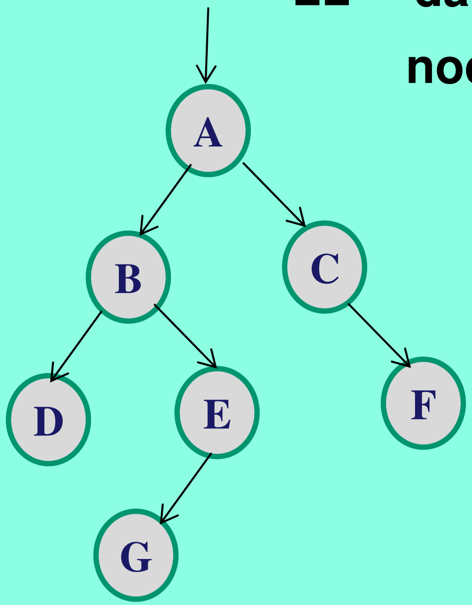
Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.



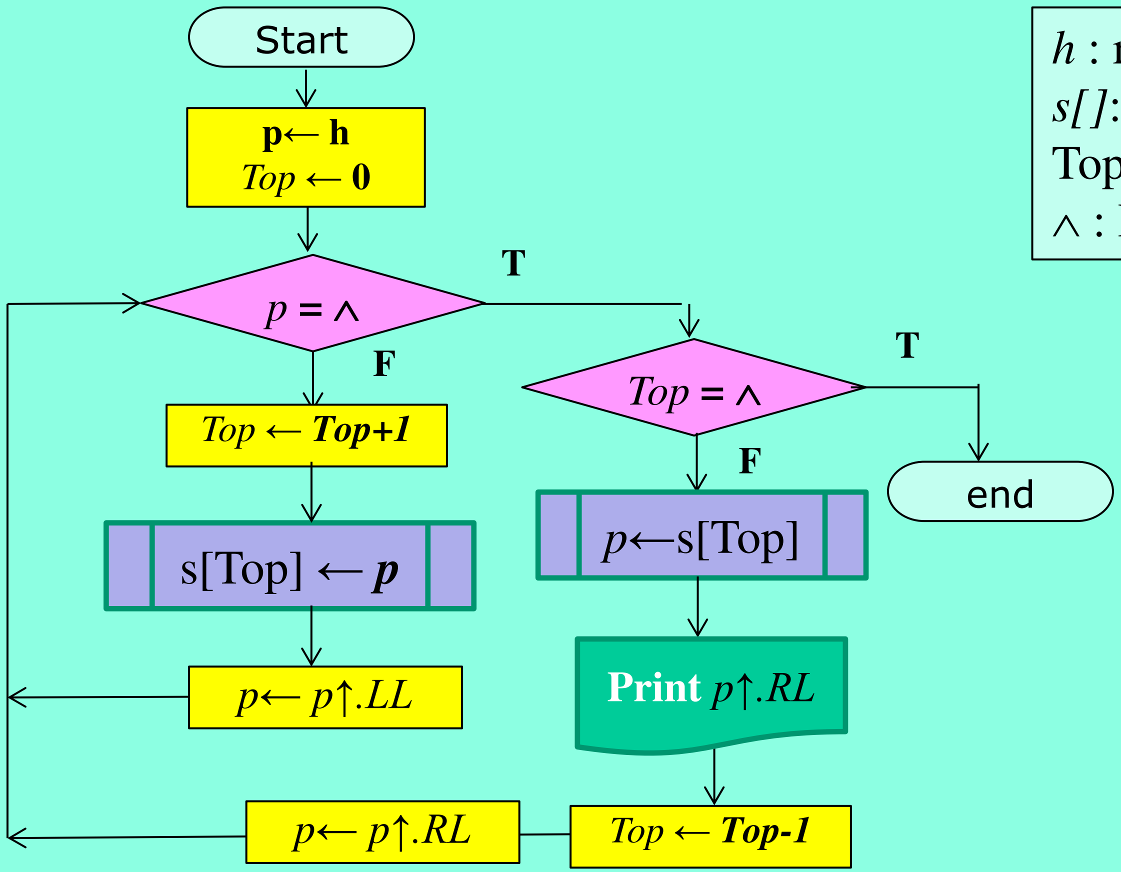
h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL



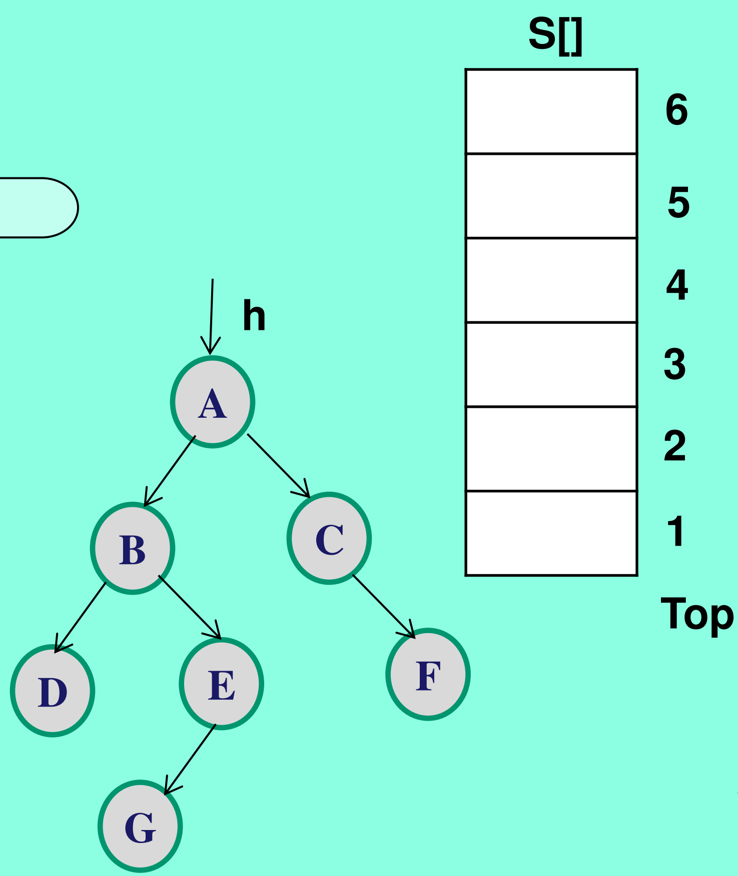


Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

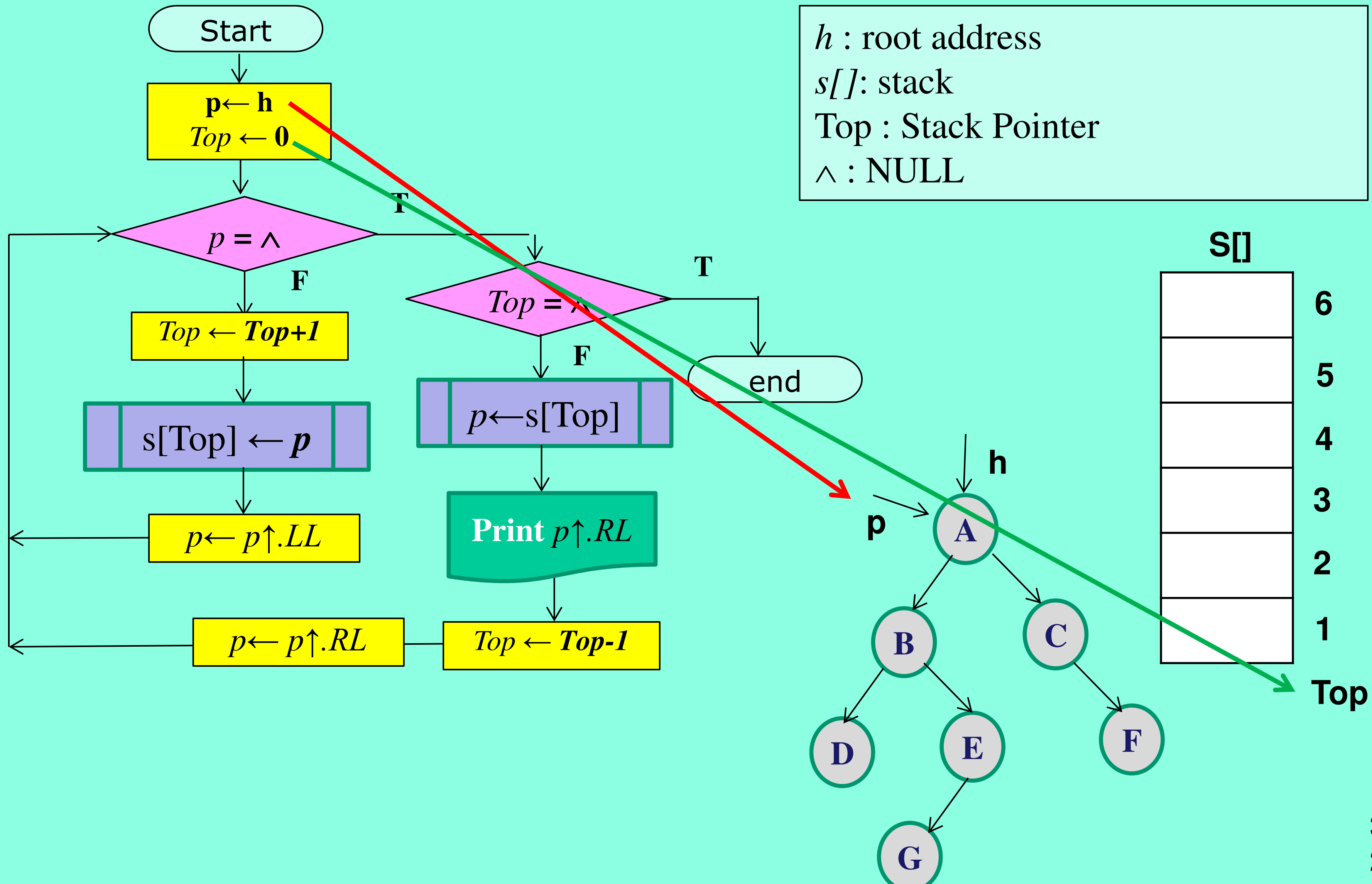


h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL



Traversing BT

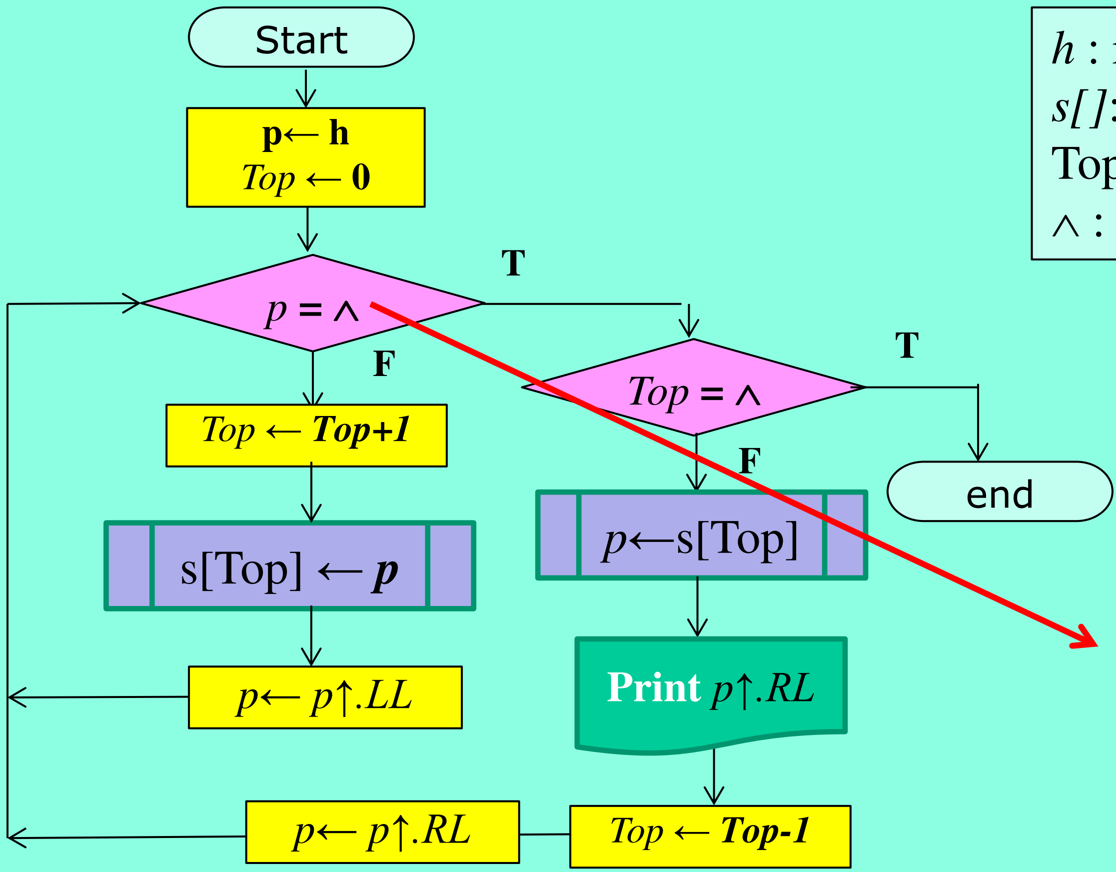
Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

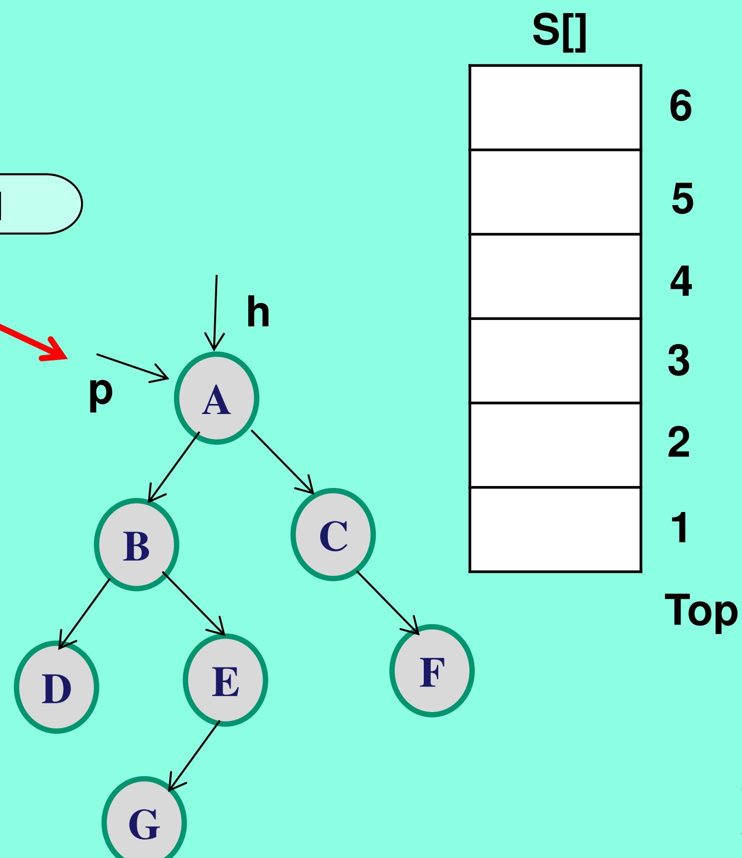


Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

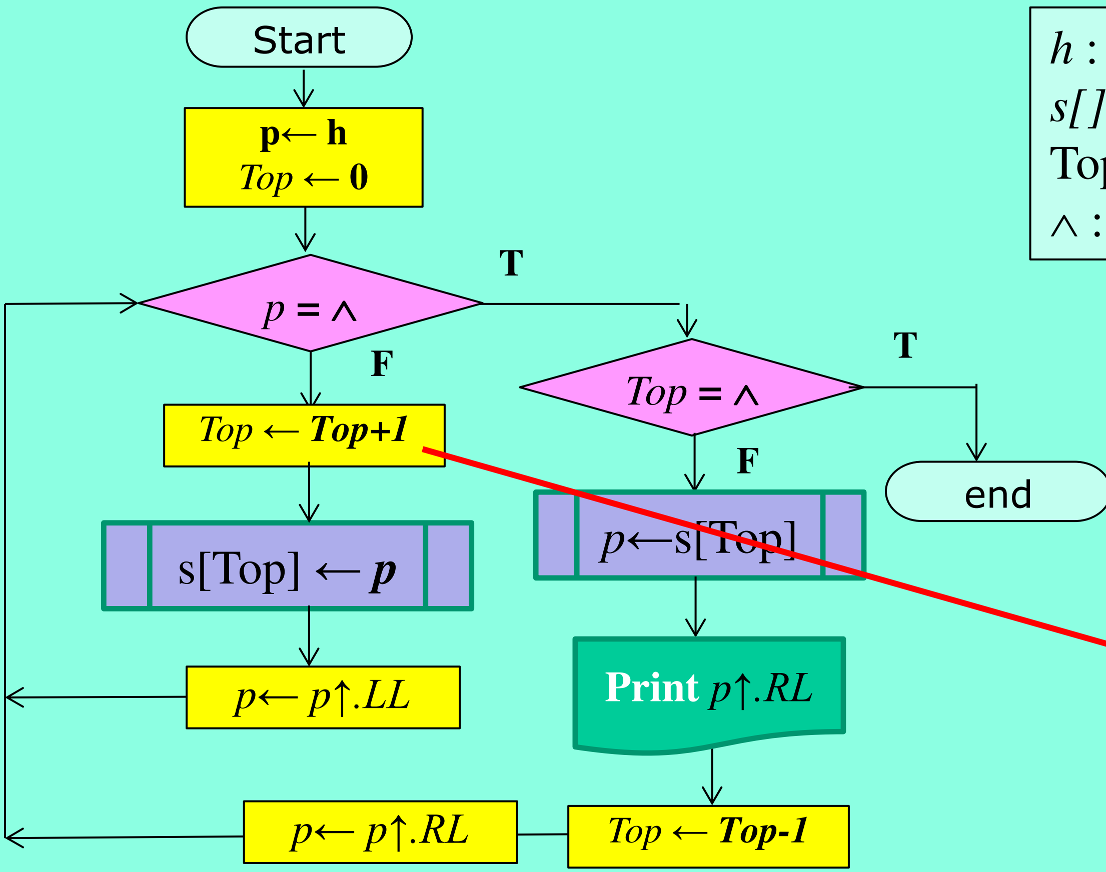
h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL



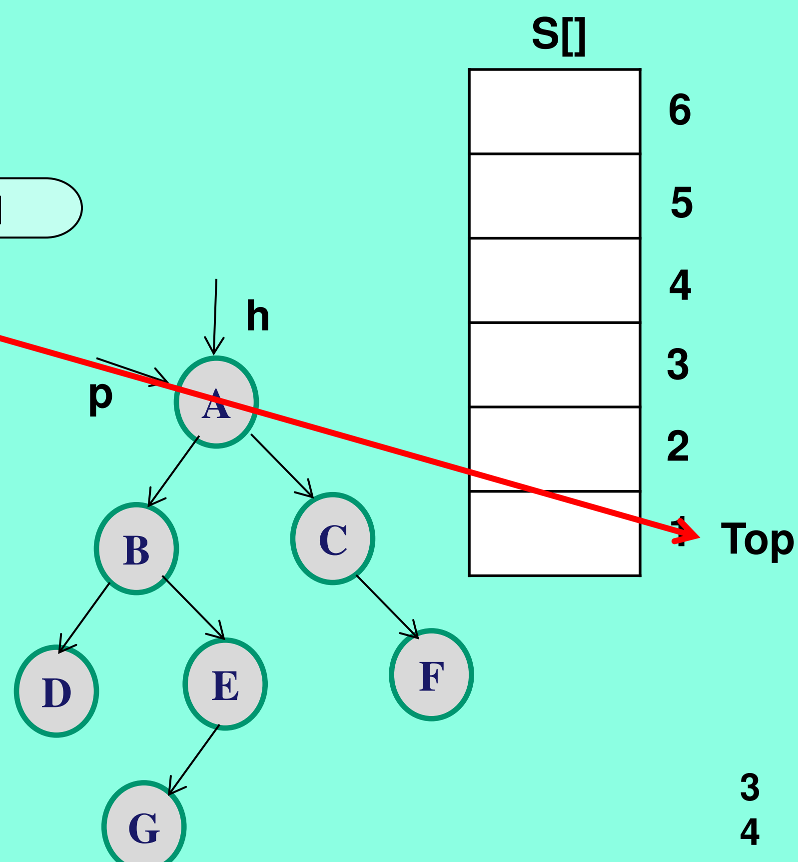


Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

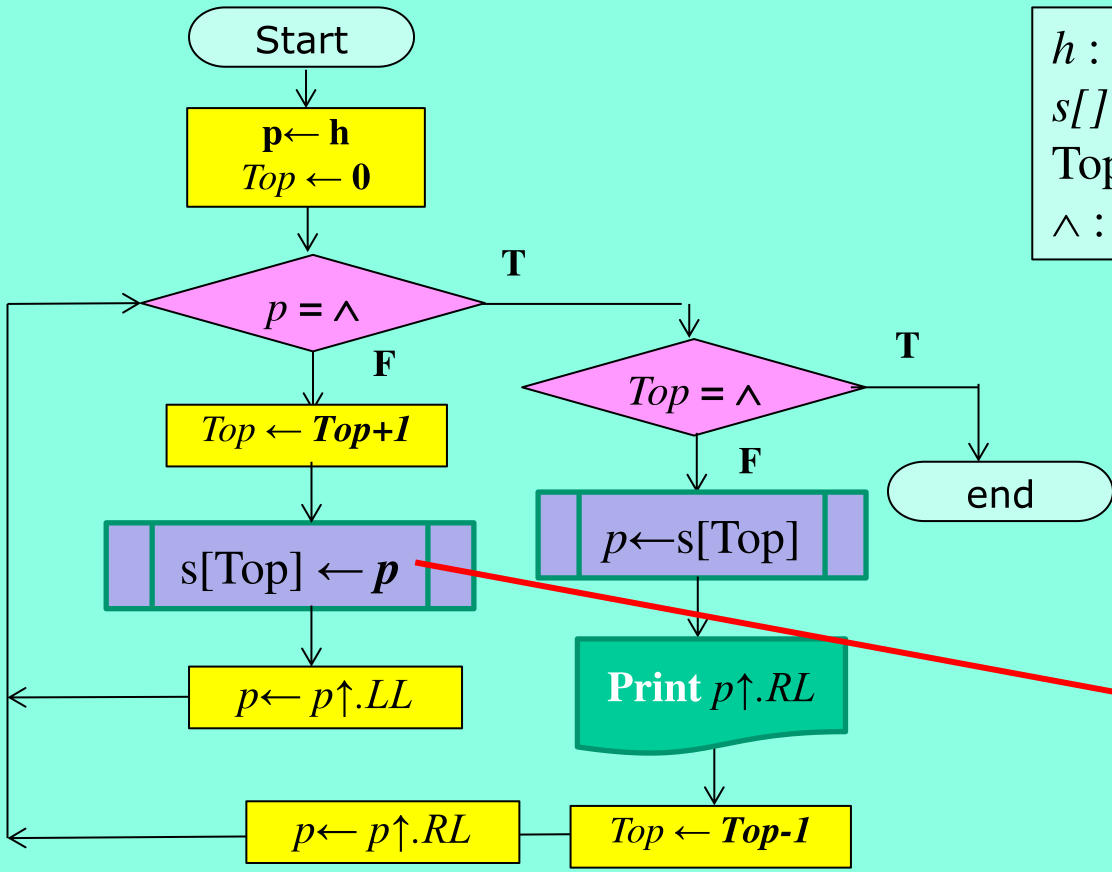


h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL

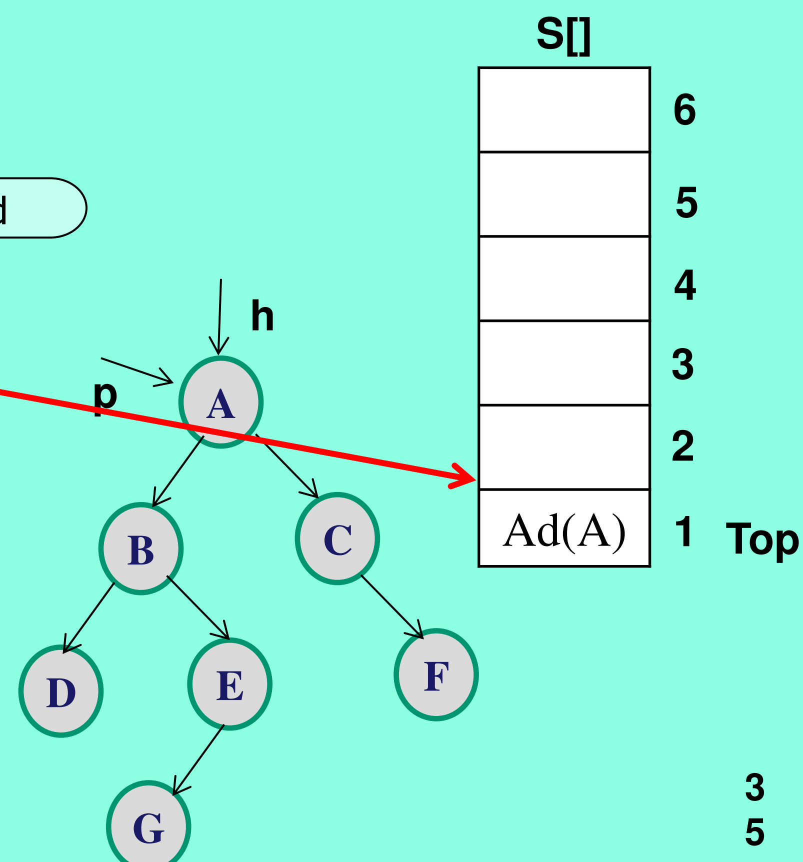


Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

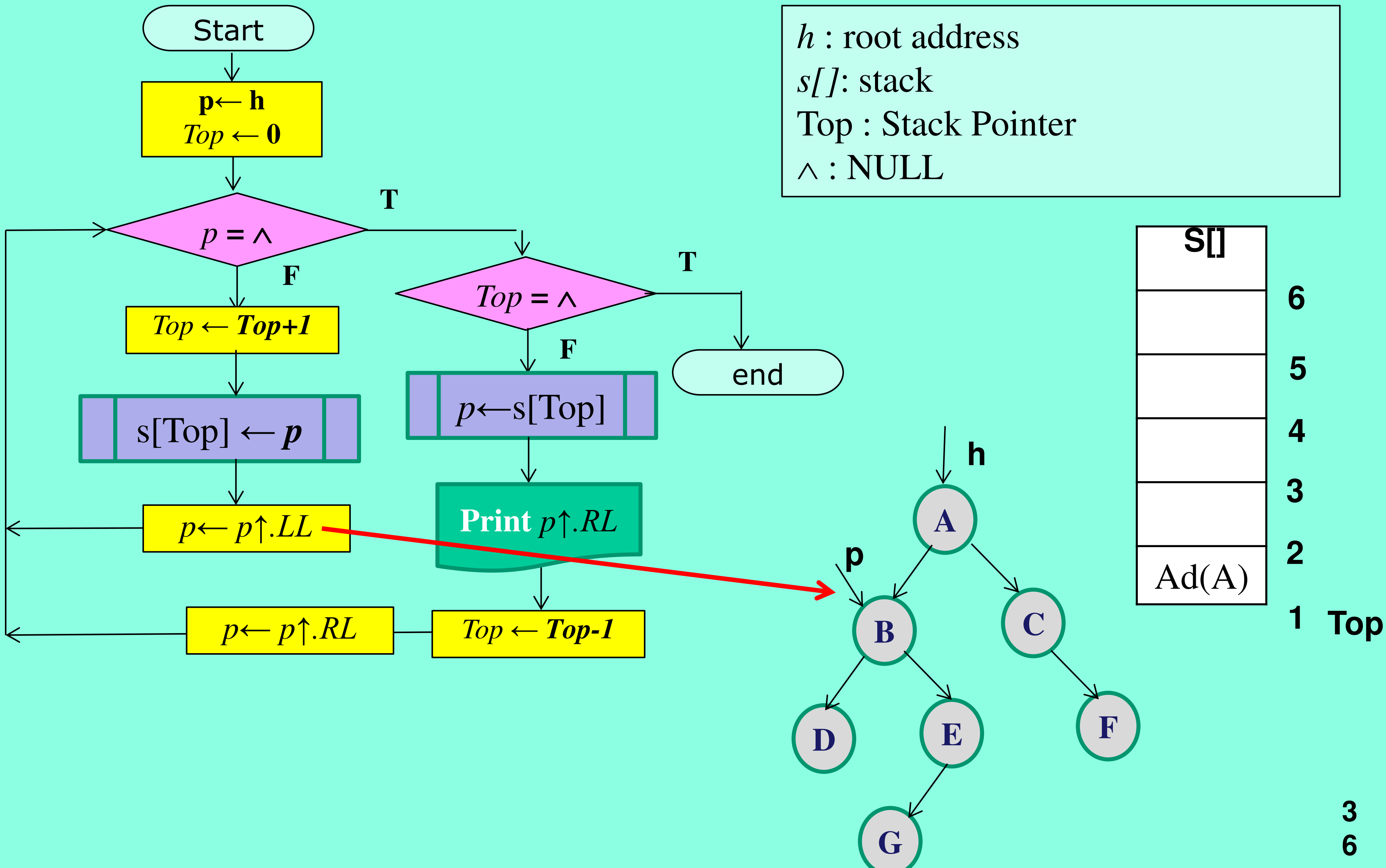


h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL



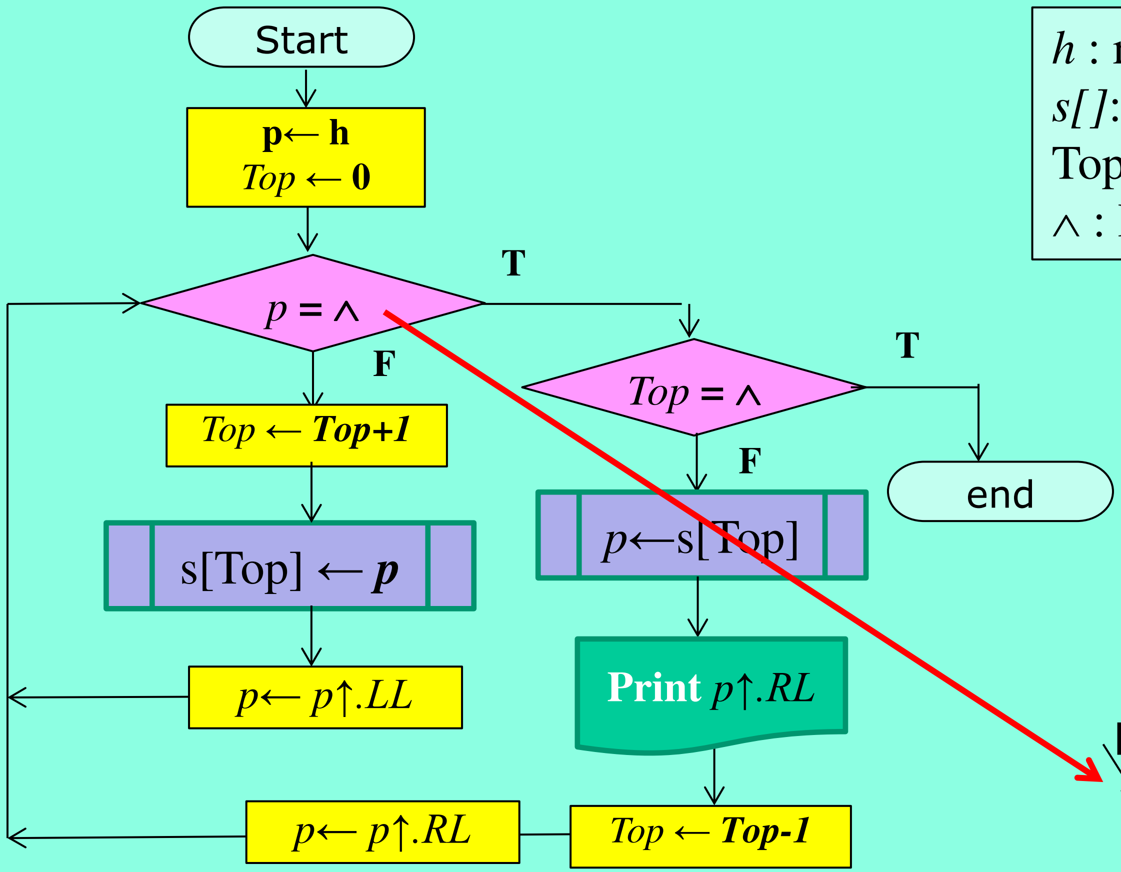
Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

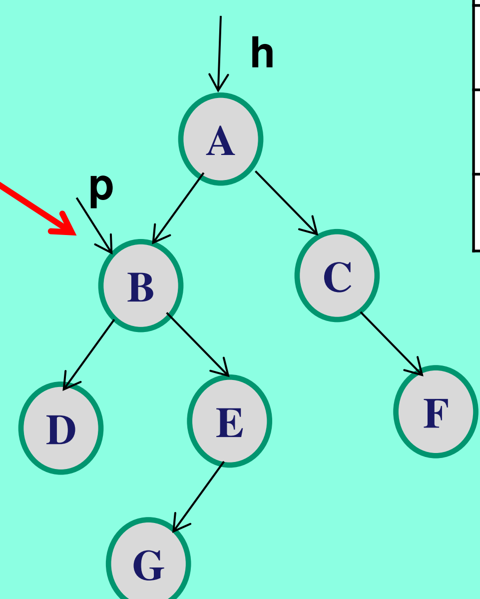


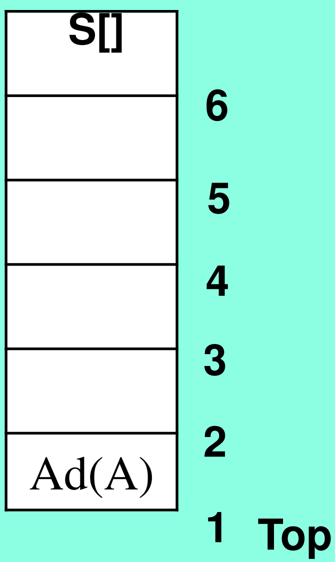
Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.



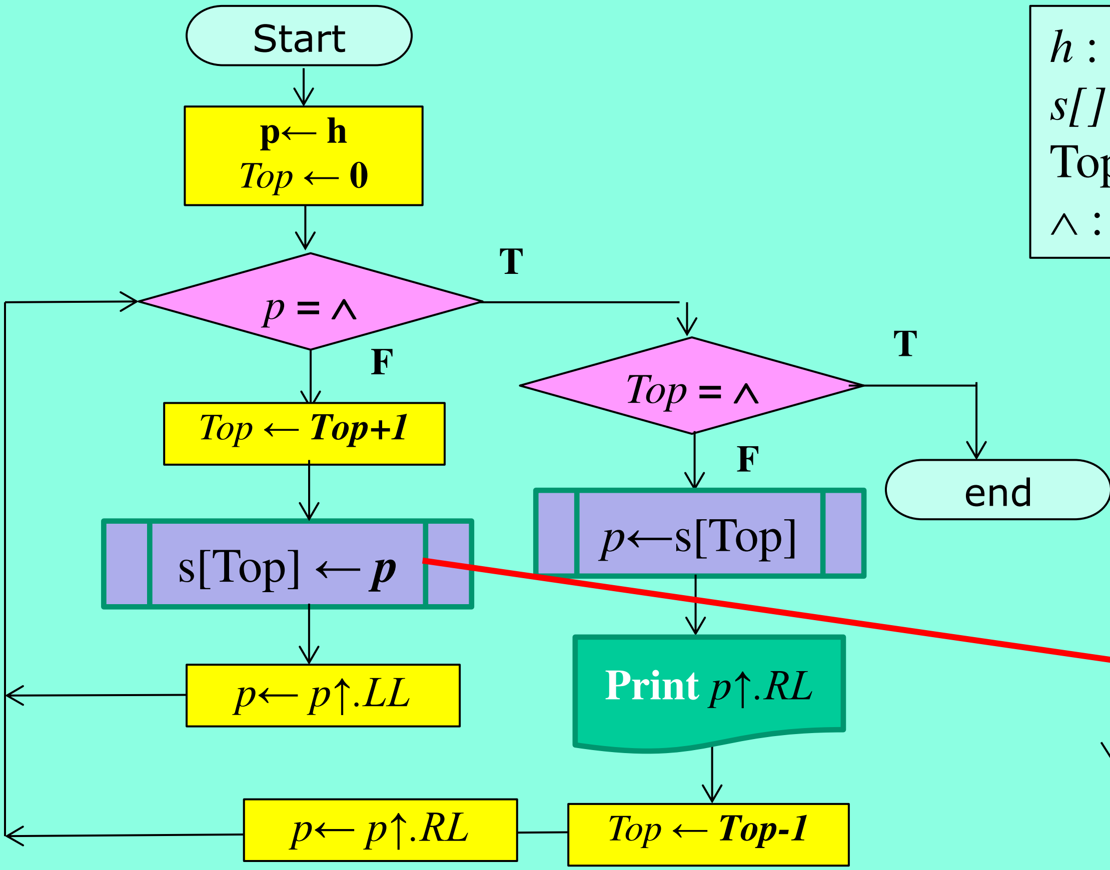
h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL



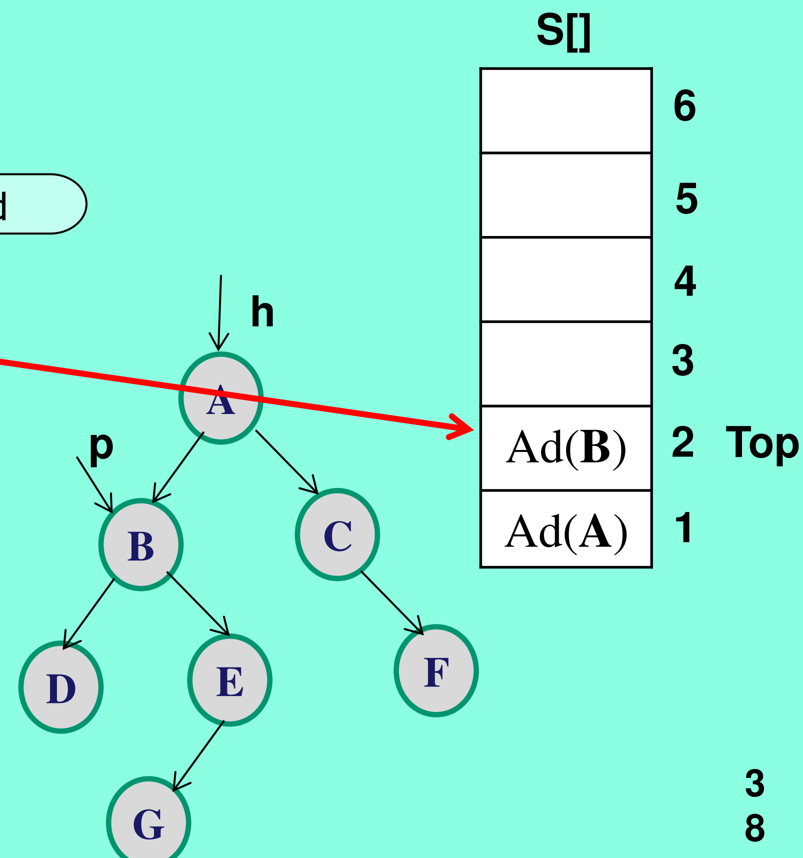


Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.



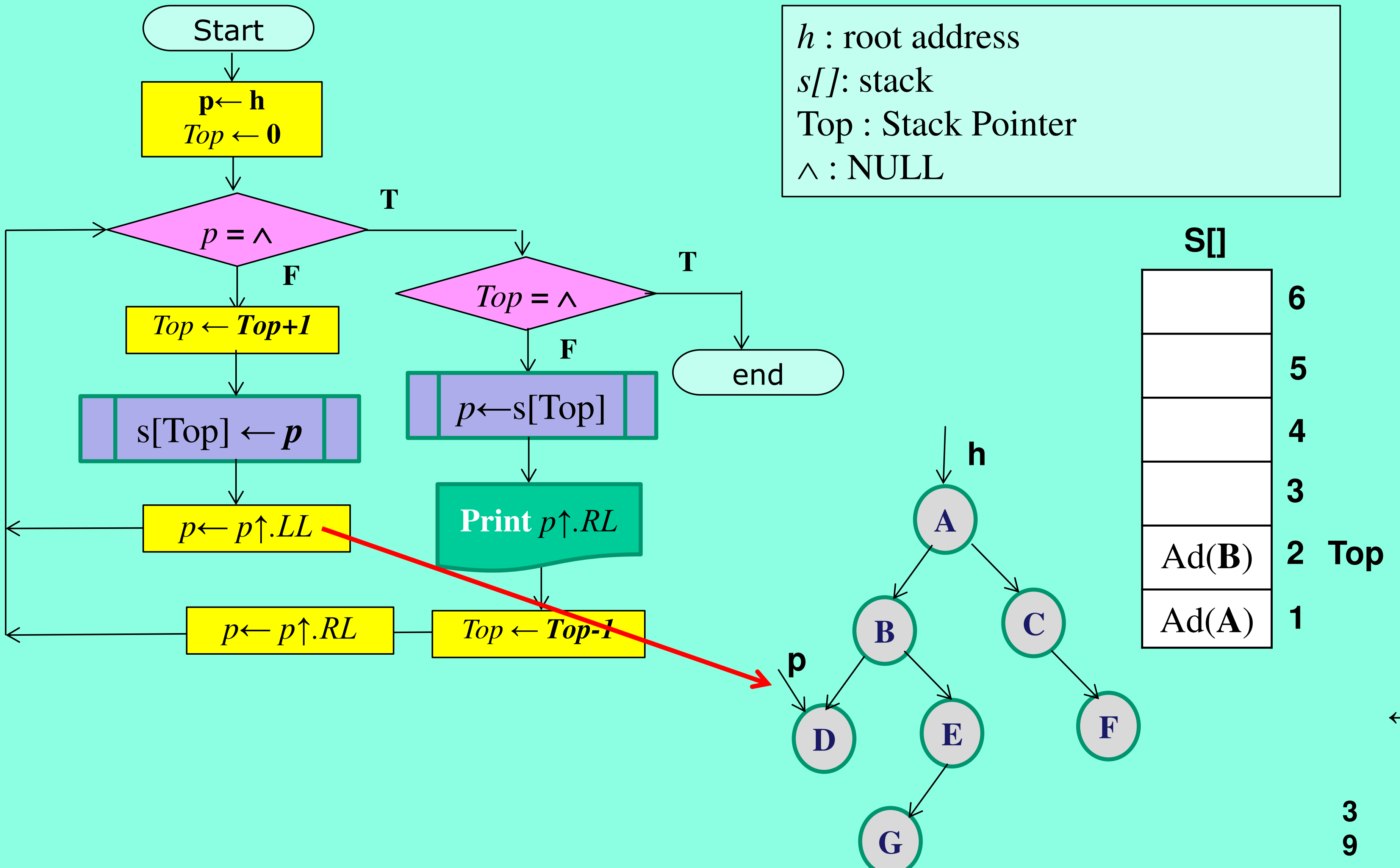
h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL



Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

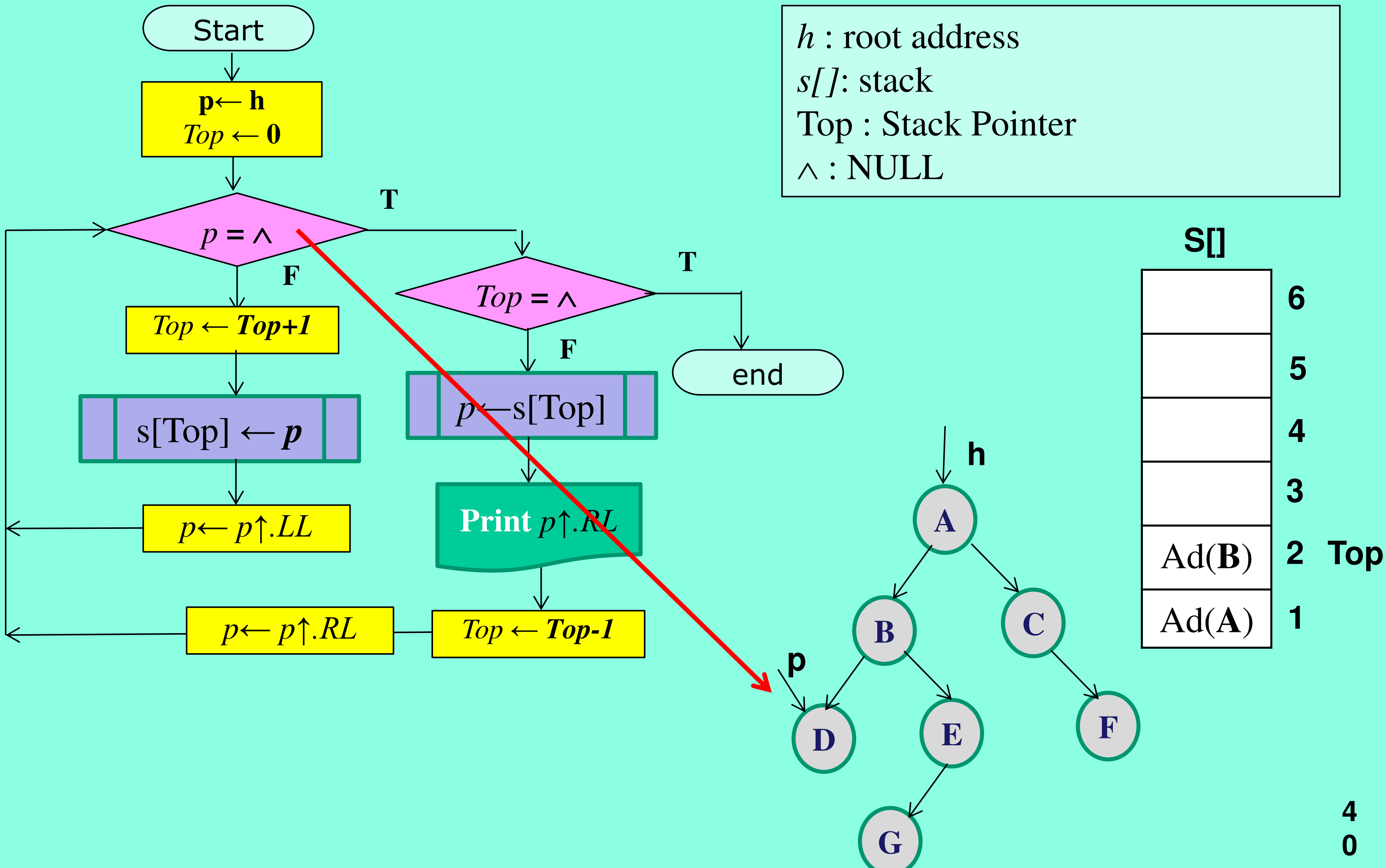
h : root address
s[ ]: stack
Top : Stack Pointer
$\wedge$: NULL



Traversing BT

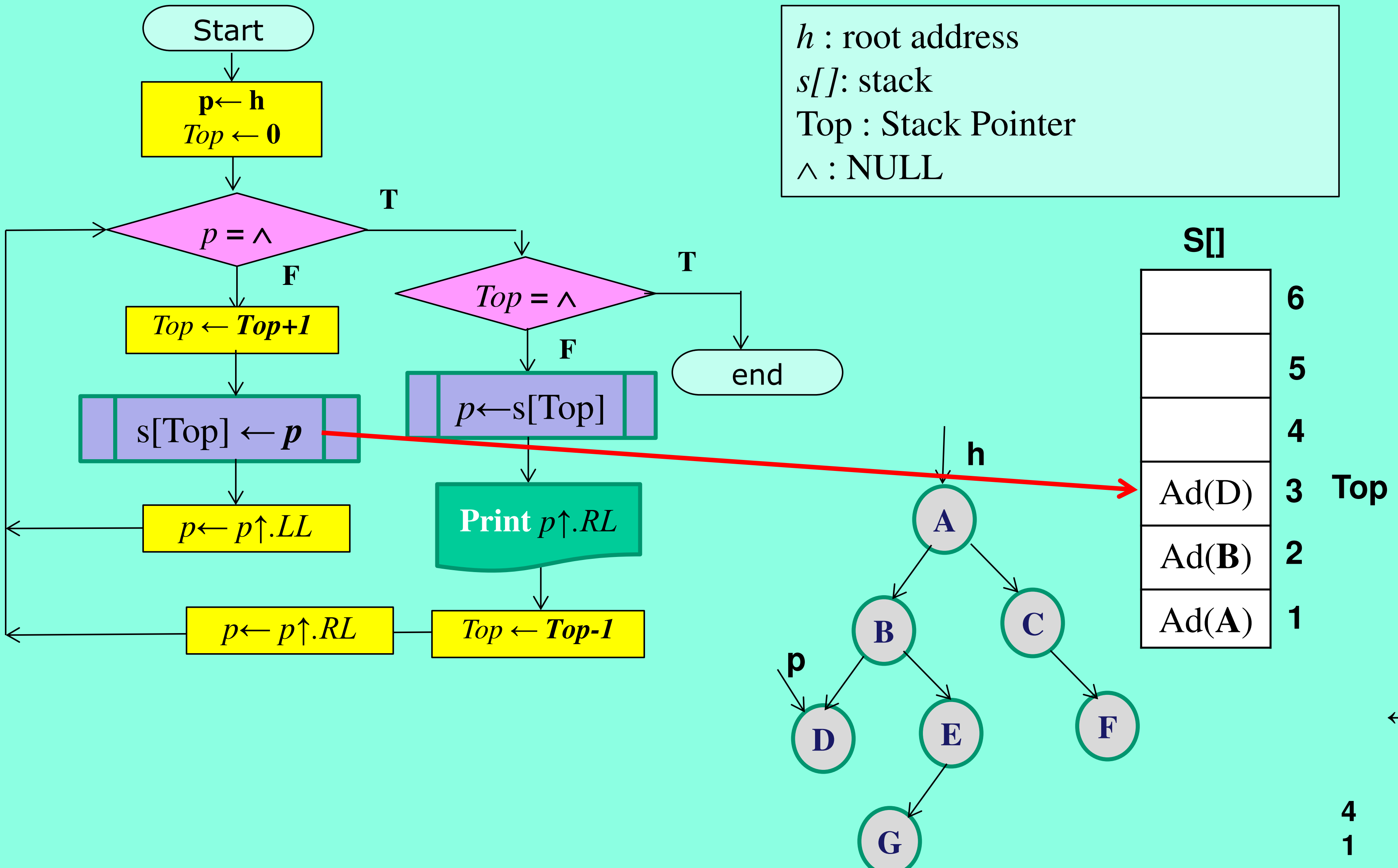
Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL



Traversing BT

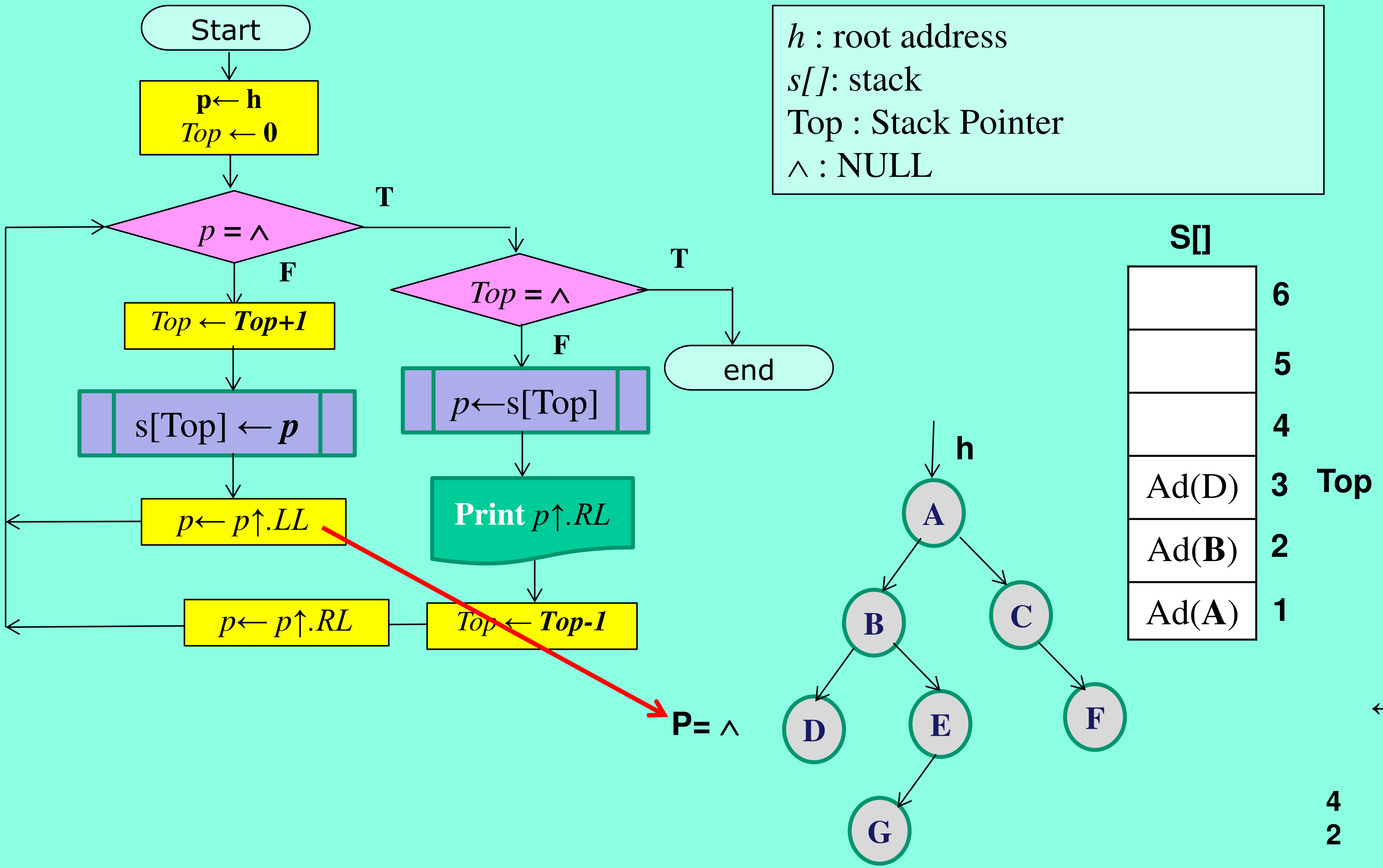
Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.



Traversing BT

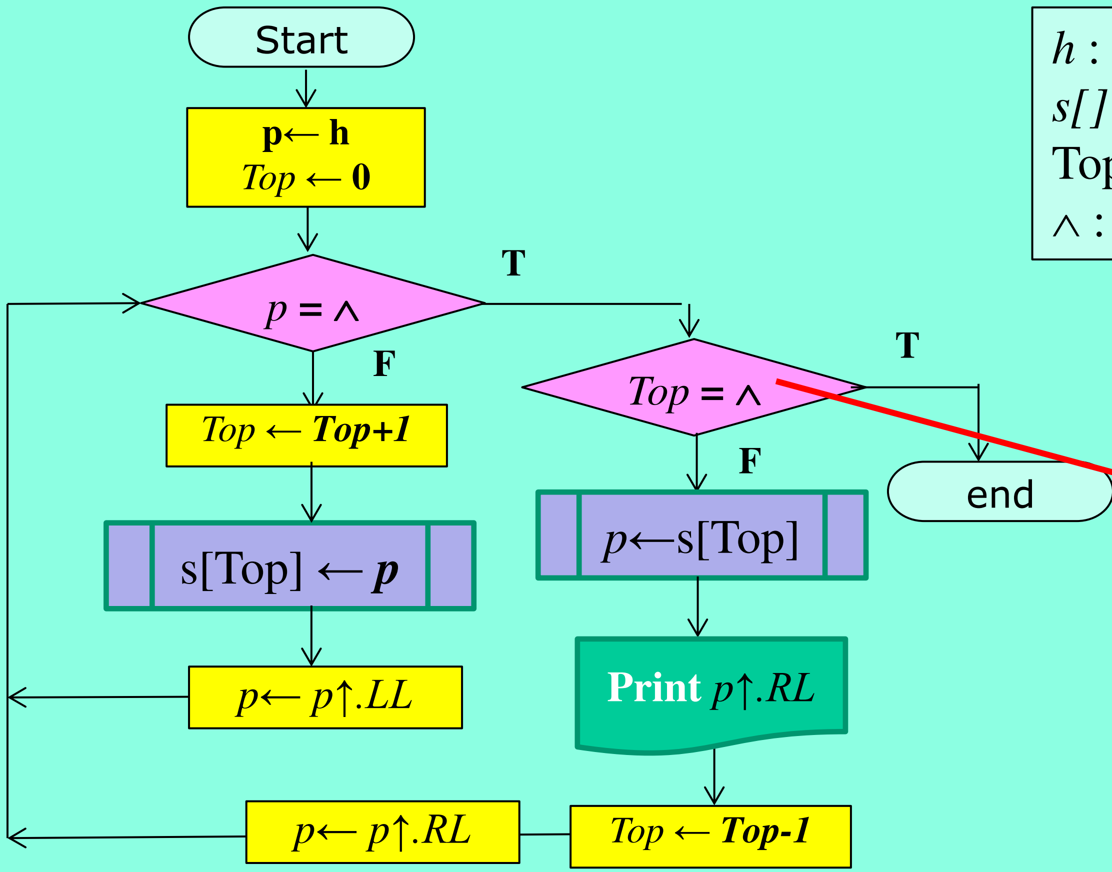
Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL

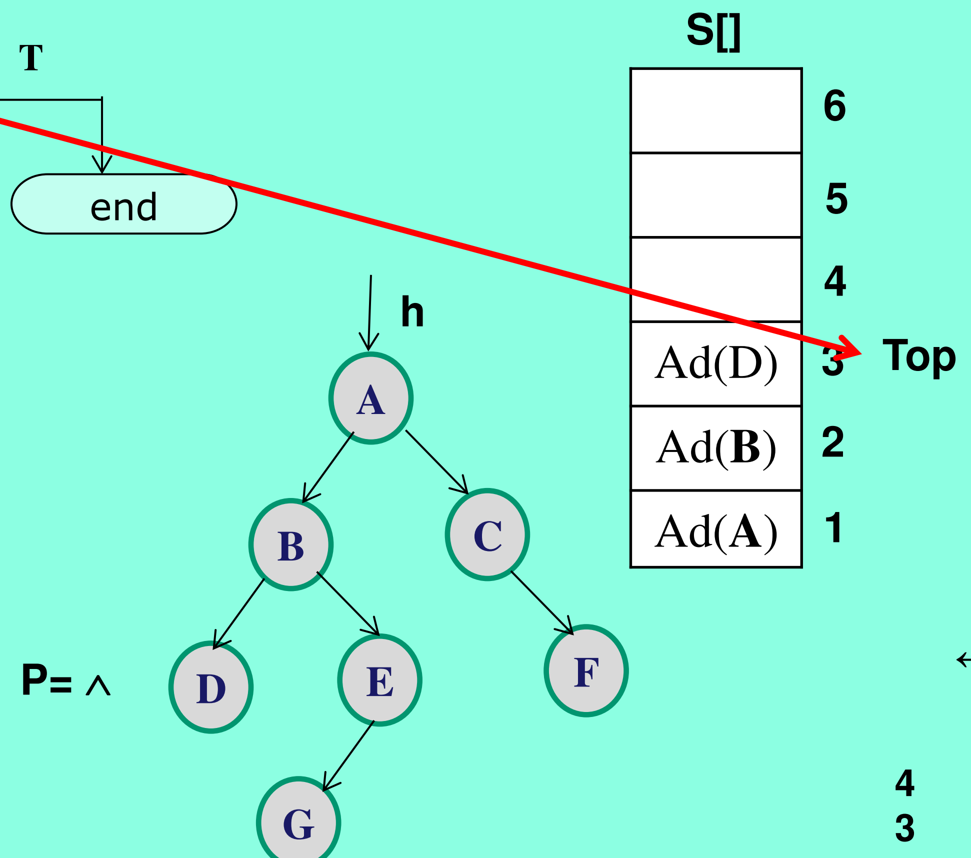


Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

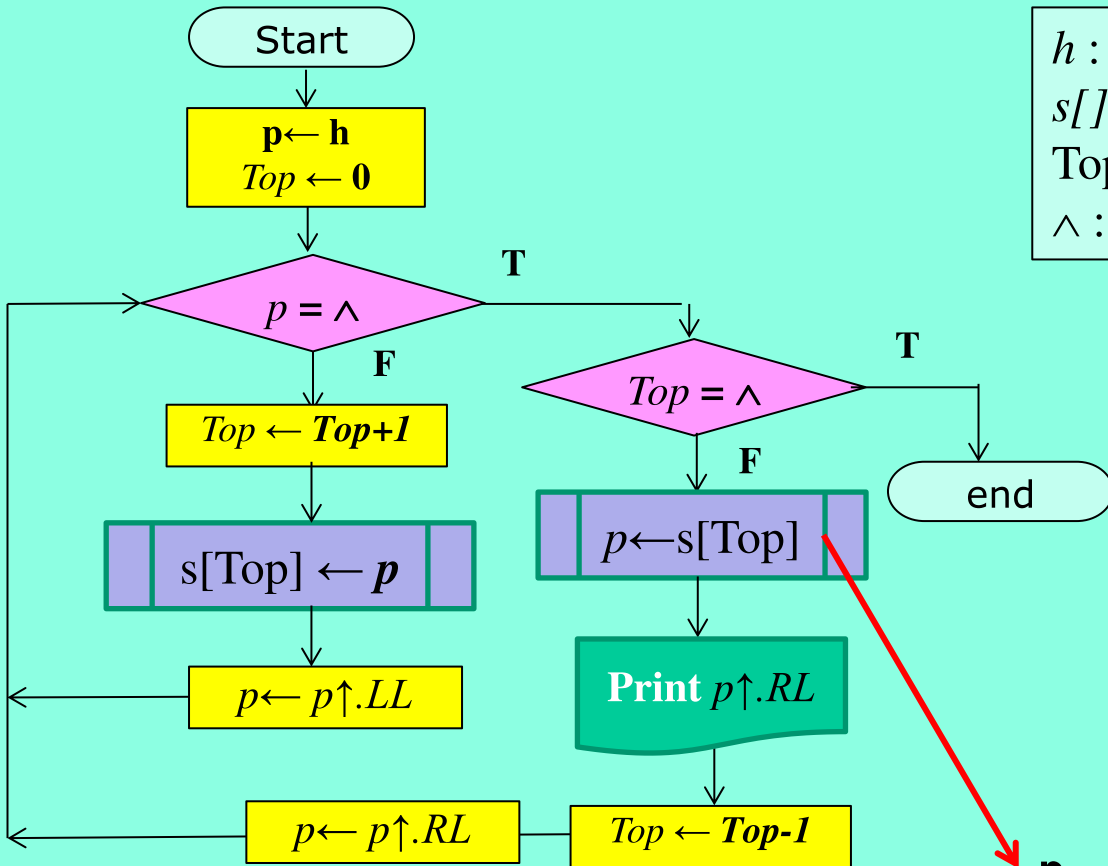


h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL

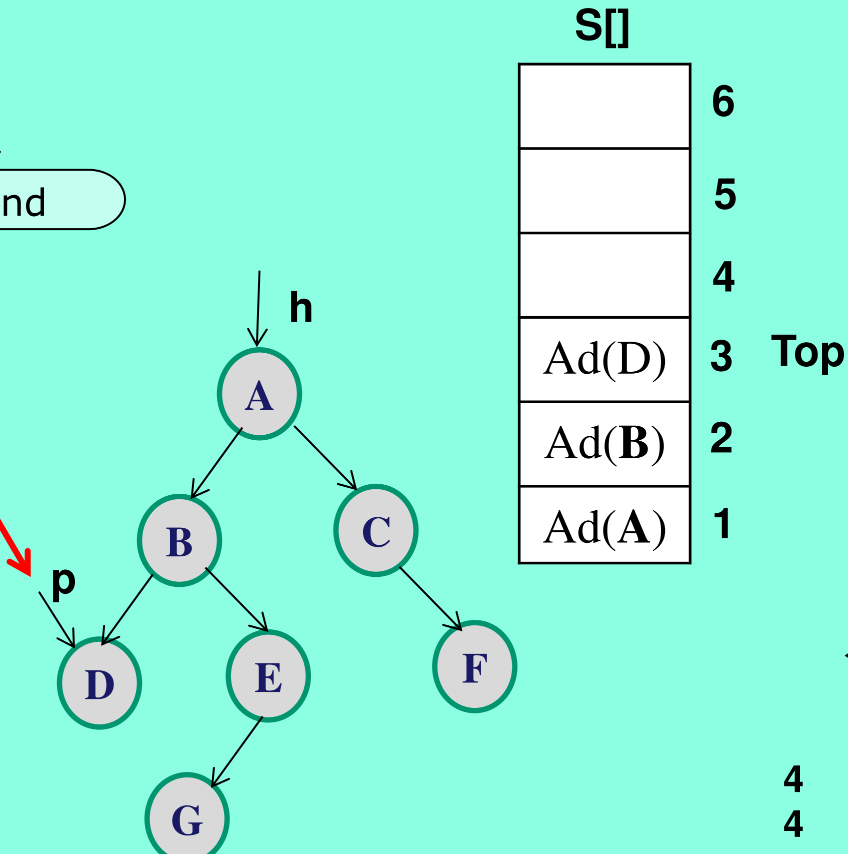


Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

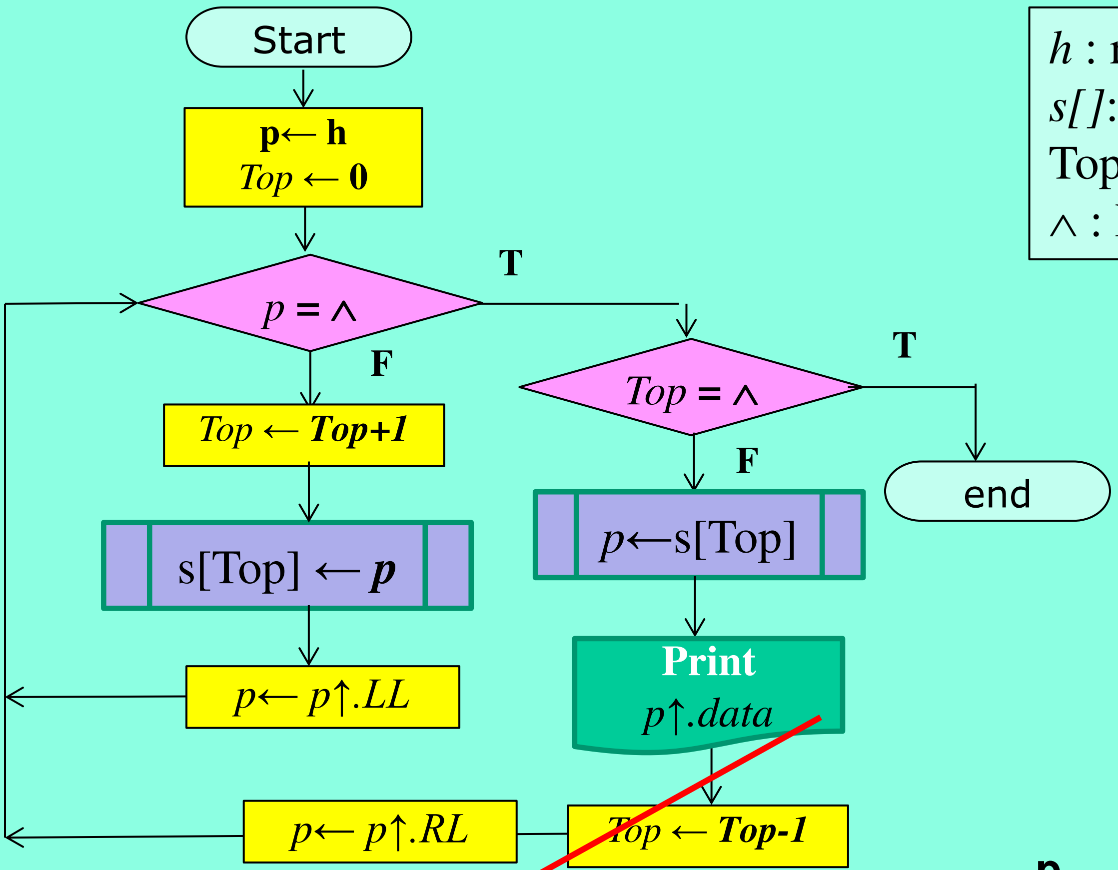


h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL

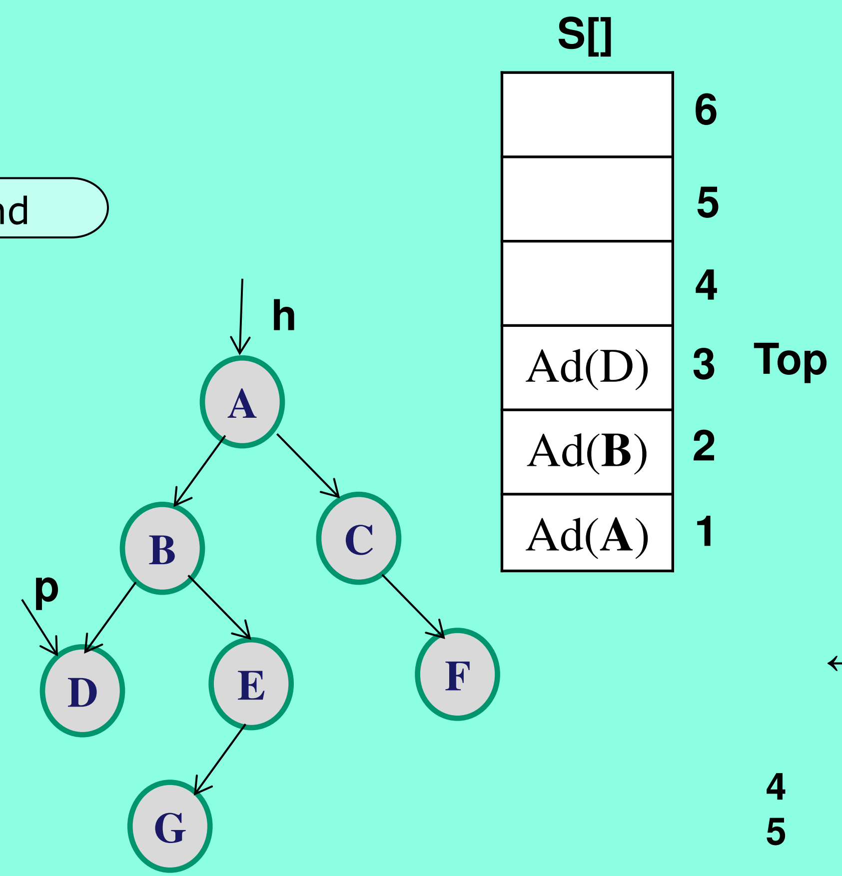


Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.



h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL



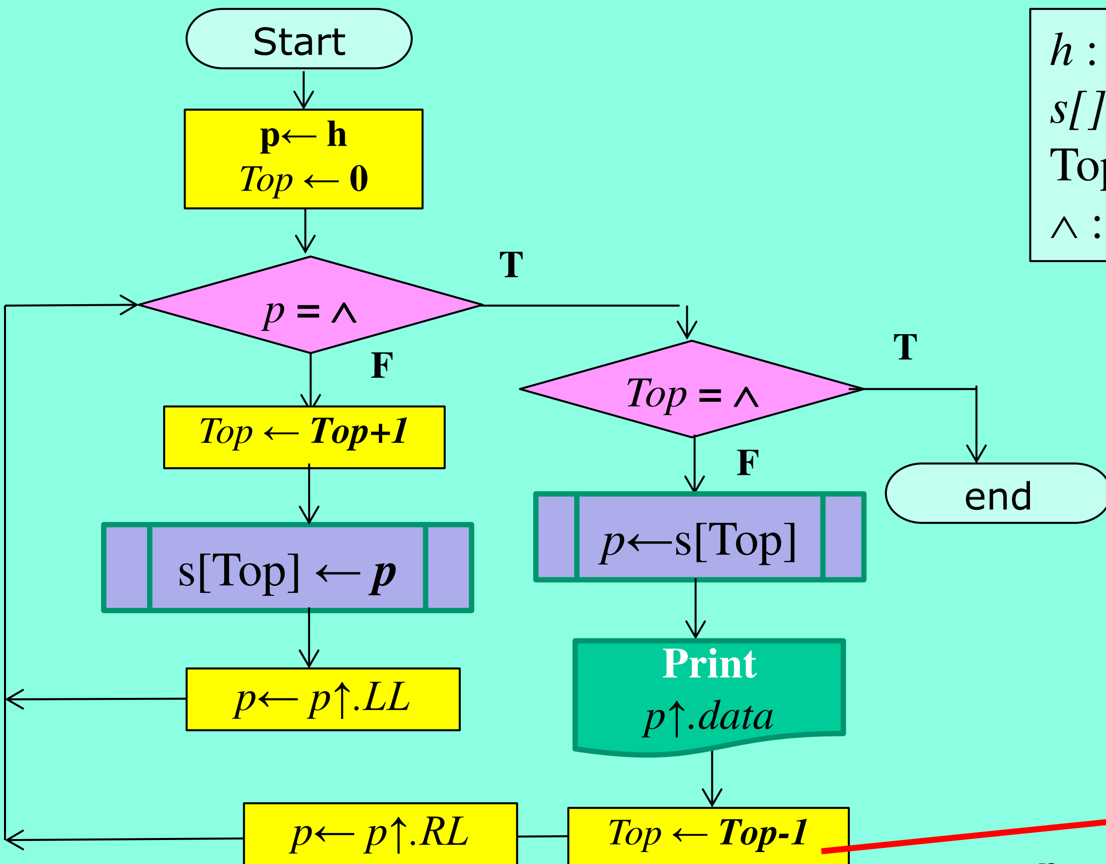
Output

D

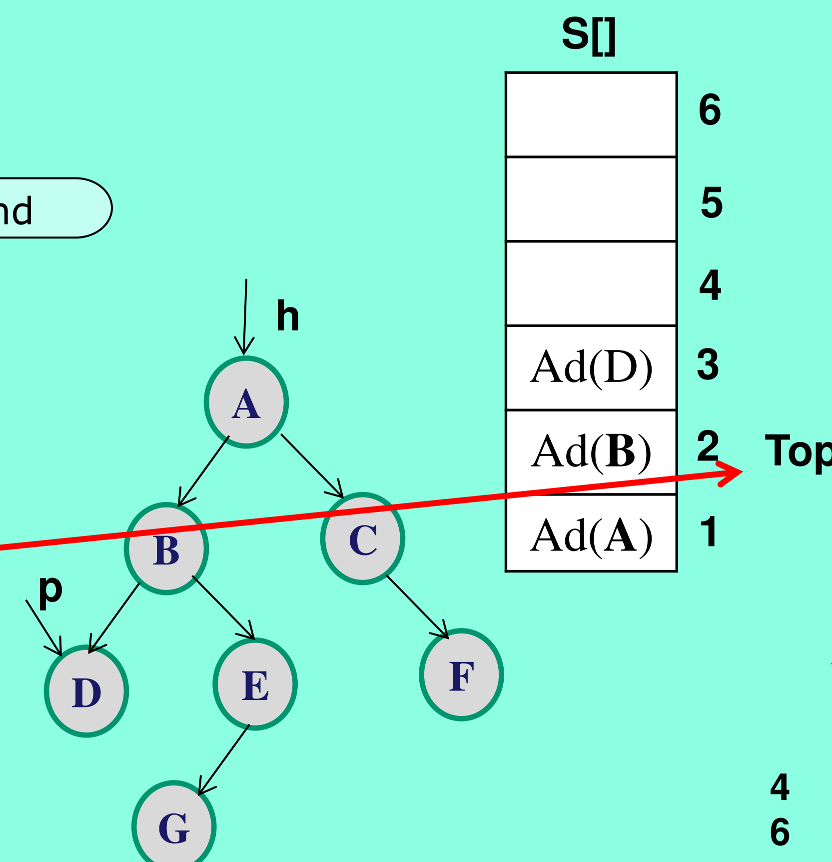
4
5

Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.



h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL

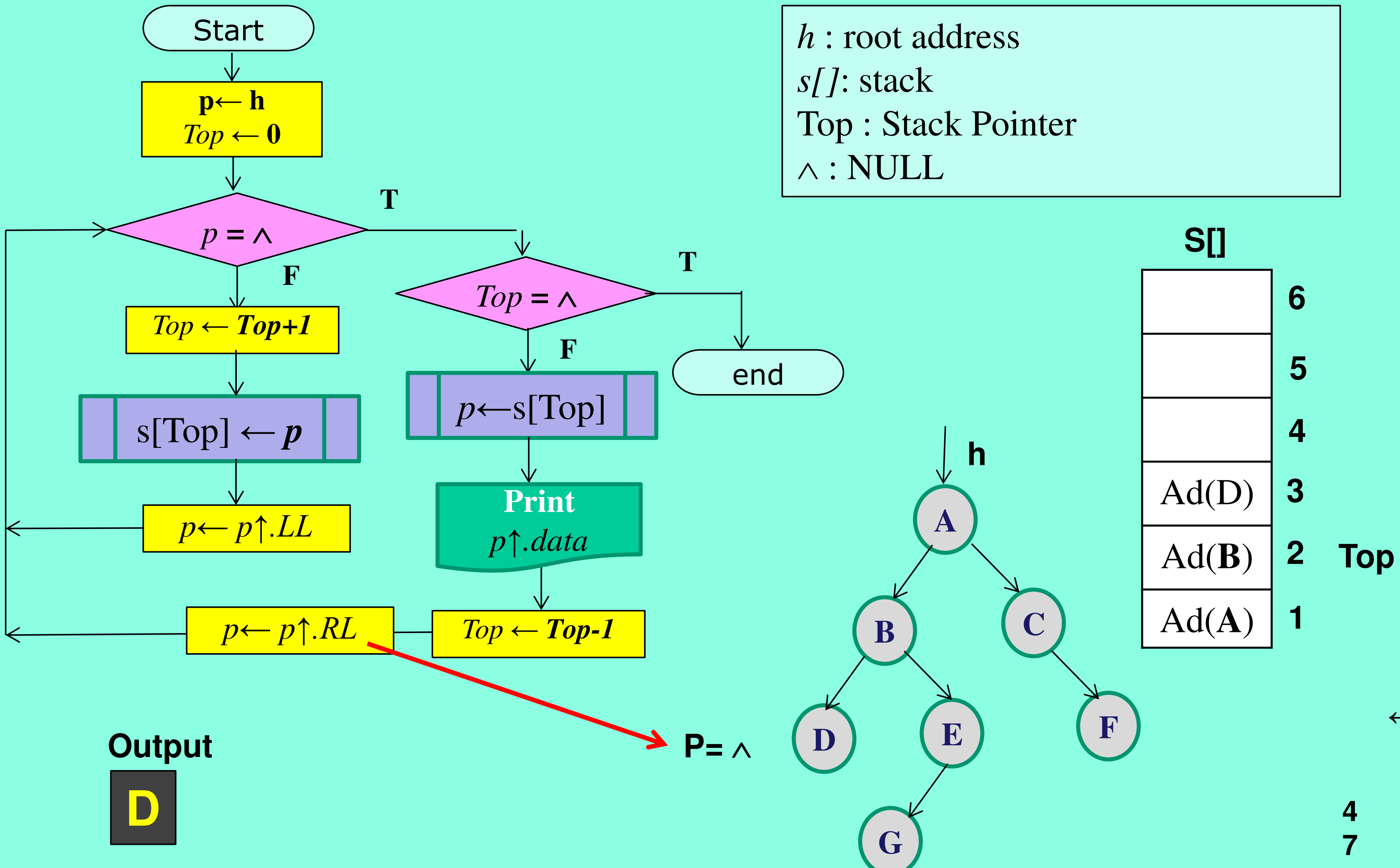


Output

D

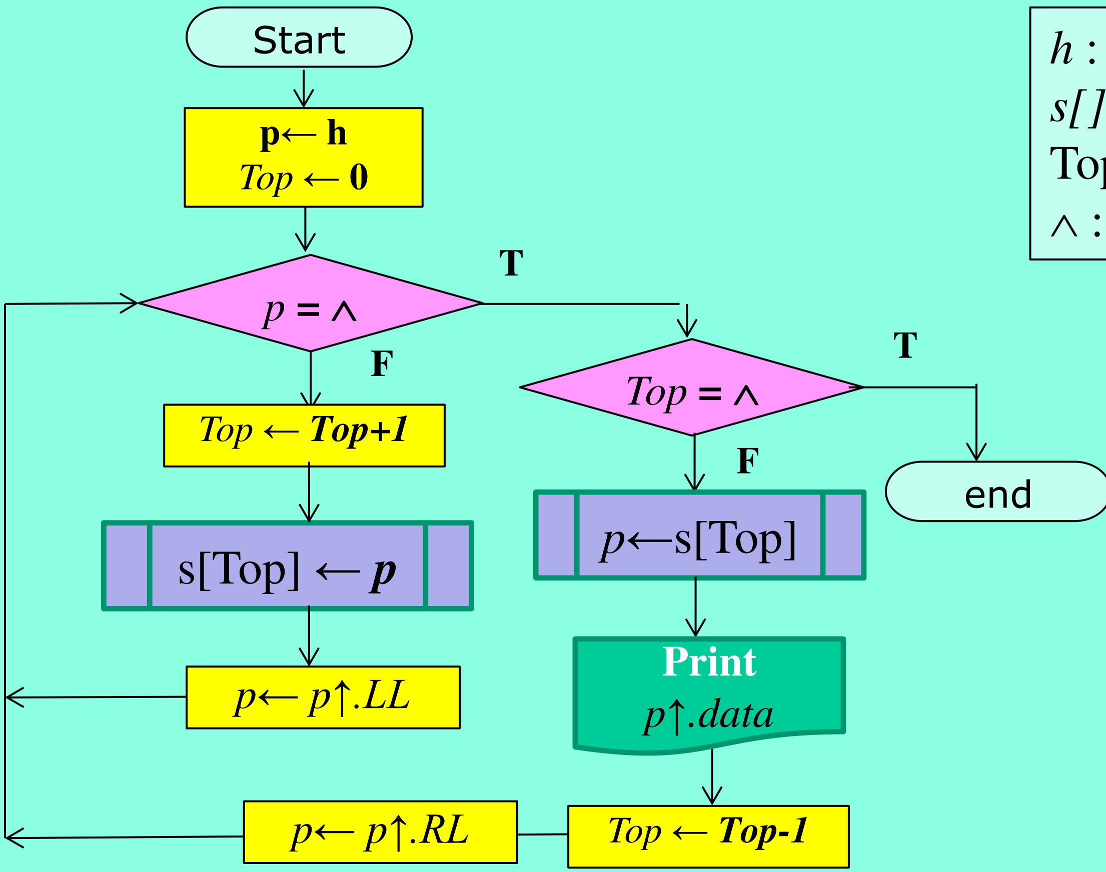
Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.

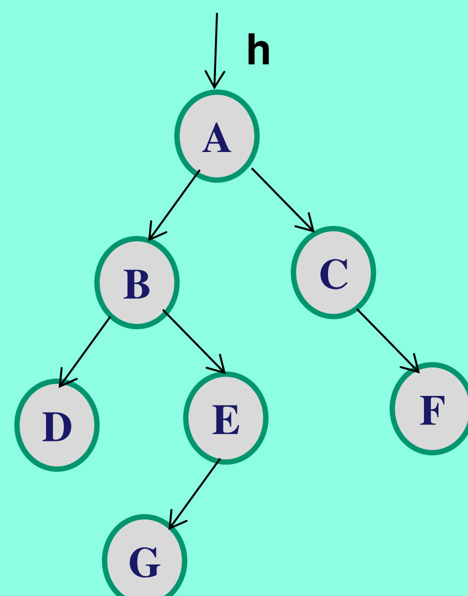


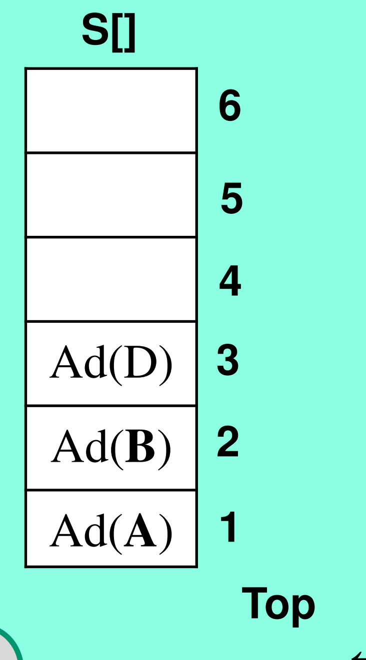
Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in inorder traversing.



h : root address
s[]: stack
Top : Stack Pointer
∧ : NULL



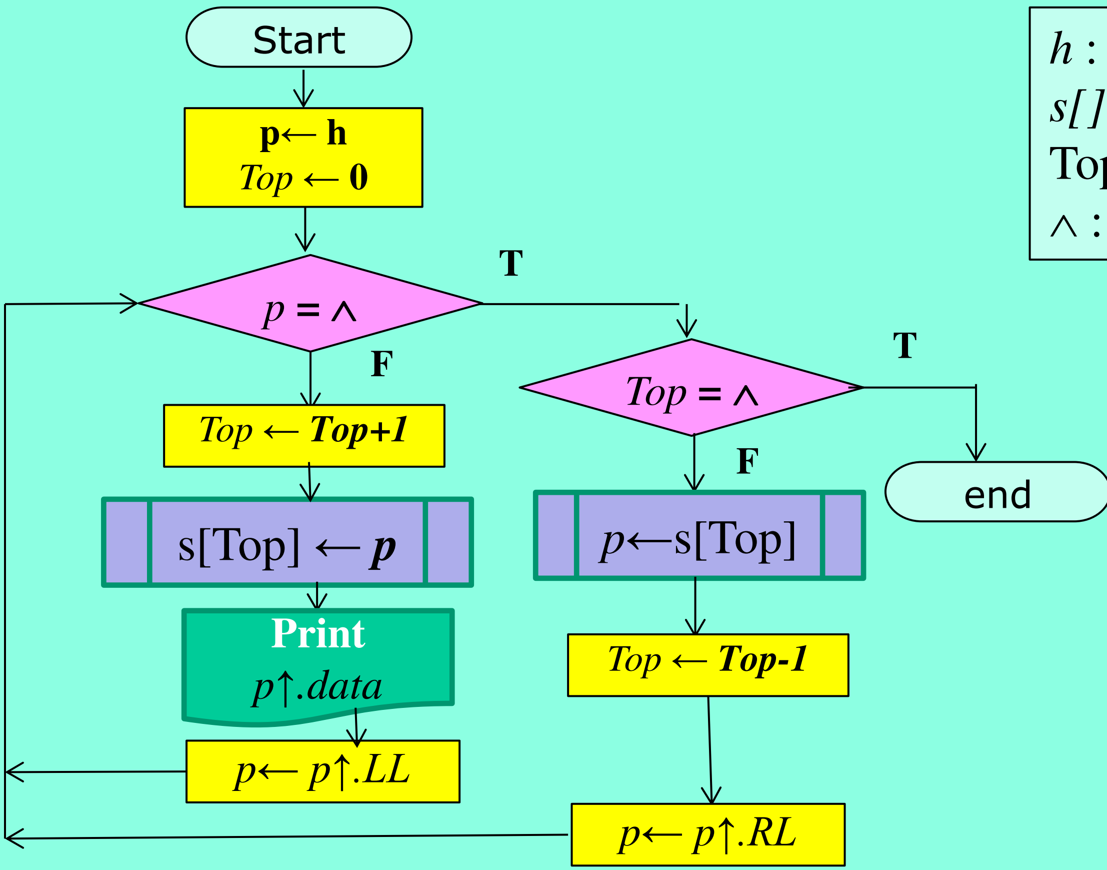


Output

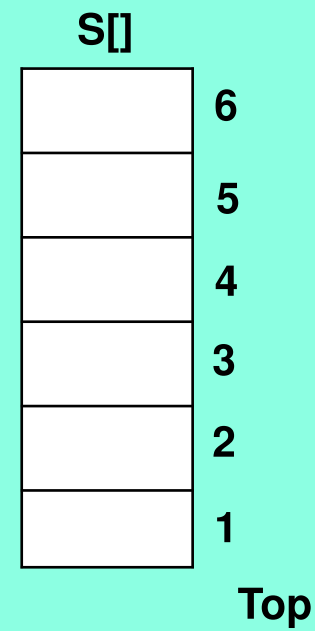
D B G E A C F

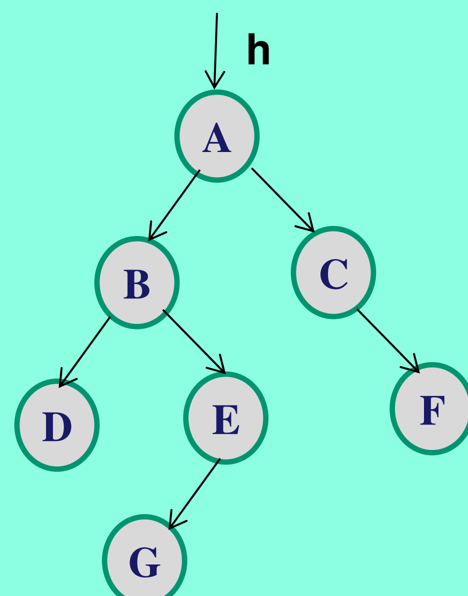
Traversing BT

Topic 1: Write an Algorithm to traverse a binary tree in Preorder traversing.



h : root address
s[ ]: stack
Top : Stack Pointer
∧ : NULL





Output

A B D E G C F

Construct BT using Inorder & Preorder Traversing

Consider the following example:

in-order: 4 2 5 (1) 6 7 3 8

pre-order: (1) 2 4 5 3 7 6 8

From the pre-order sequence, we know that first element is the root. We can find the root in in-order sequence. Then we can identify the left and right sub-trees of the root.

For this example, the constructed tree is