Tree

The depth of a binary tree is the maximum level of any leaf in the tree. This equals the length of the longest path from root to any leaf.

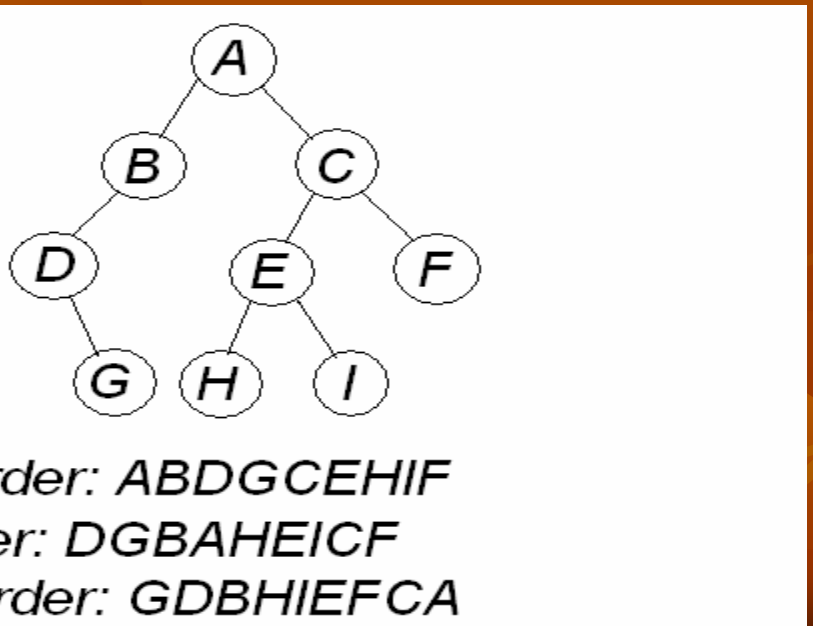
A binary tree of depth d is a complete binary tree if :

- Any node at level than d-1 has 2 sons
- For any node in the tree with a right descendent at level d, it must have a left descendent of the node is either a leaf at level d or has 2 sons

- struct nodetype{
    int info;
    struct nodetype *left,*right;

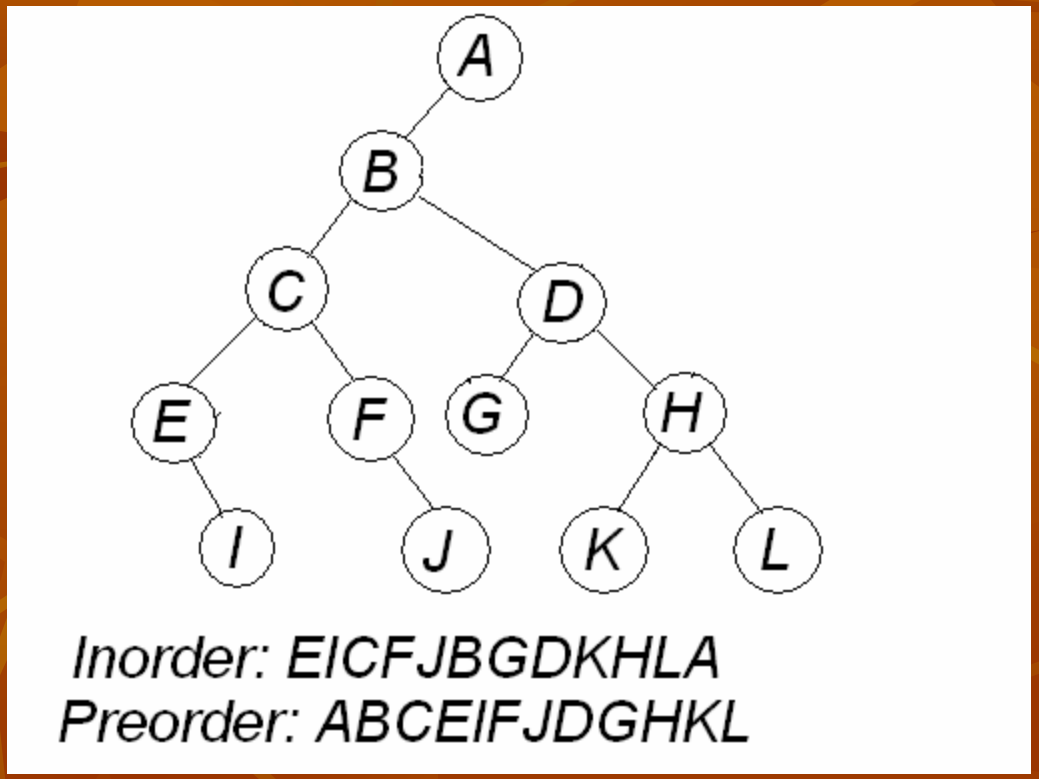
};

typedef struct nodetype *NODEPTR ;



```
void perorder(NODEPTR tree)
{
        if(tree!=NULL){
                printf("%d \n",tree->info);
                preorder(tree->left);
                preorder(tree->right);
}/* end preorder*/
```

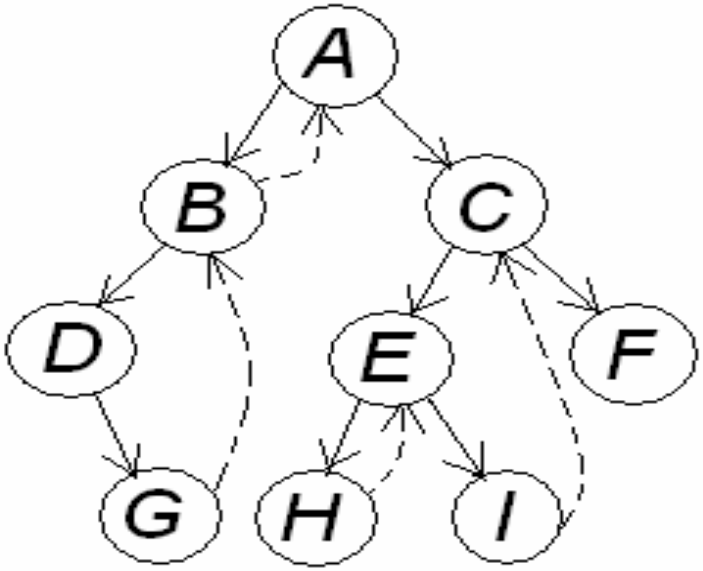
```
void inorder (NODPTR tree)
{           struct stack{
                int top;
                NODEPTR item [MAXSTACK];
            }s;
            NODEPTR p;
            s.top=-1;
            p=tree;
            do{
                        while (p!=NULL){
                                    push (s,p);
                                    p=p->left;
                        }/* end while*/
                        if (!empty(s)){
                                    p=pop(s);
                                    printf( “%d\n”, p->info);
                                    p=p->right;
                        }/*end if*/
            }while (!empty(s) || p!=NULL);
}/*end inorder */
```



Inorder: EICFJBGDKHLA
Preorder: ABCEIFJDGHKL

Threaded Binary Trees

```
struct nodetype{
        int info;
        struct nodetype *left;
        struct nodetype *right;
        int rthread
}
   /* rthread is TRUE if right is NULL or*/
   /* a non NULL thread */
typedef struct nodetype *NODEPTR ;
```


A
B
C
D
E
F
G
H
I

Right Inorder-Threaded Binary Trees

```
NODEPTR maketree (int x)
{       NODEPTR p;
        p=getnode();
        p->info=x;
        p->left=NULL;
        p->right=NULL;
        p->rthread=TRUE;
        return p;
}/* end maketree */
```

```
void setleft (NODEPTR p, int x)
{       NODEPTR q;
        if (p==NULL)
                printf (“void insertion \n”);
        else if (p->left != NULL)
                printf (“invalid insertion \n”);
              else {
                q=getnode();
                q->info=x;
                p->left=q;
                q->left=NULL;
                q->right=p;
                q->rthread=TRUE;
              }/* end else */
}/* end setleft */
```

```
void setright (NODEPTR p, int x)
{       NODEPTR q, r;
        if (p==NULL)
                printf ("void insertion \n");
        else if (!p->rthread)
                printf ("invalid insertion \n");
             else {
                  q=getnode();
                  q->info=x;
                   r=p->right;
                   p->right=q;
                   p->rthread=FALSE;
                   q->left=NULL;
                   /* The inorder successor of node(q) is */
                   /* the previous successor of node (p) */
                   q->right=r;
                   q->rthread=TRUE;
                 }/* end else */
}/* end setright */
```

```
void intrav (NODEPTR tree)
{       NODEPTR p, q;
        p=tree;
        do{     q=NULL;
                while(p != NULL){
                /*traverse left branch */
                        q=p;
                        p=p->left;
                }/* end while */
                if (q != NULL){
                        printf ("%d \n", q->info);
                        p=q->right;
                        while (q->rthread && p !=NULL){
                          printf ("%d \n", p->info);
                          q=p;
                          p=p->right;
                        }/* end while */
                }/* end if */
        } while (q != NULL)
}/* end intrav */
```